FORMAL METHODS ADOPTION IN THE COMMERCIAL WORLD

by

Aifheli Nemathaga

submitted in accordance with the requirements for the degree of

MASTER OF SCIENCE

in the subject of

Computing

at the

UNIVERSITY OF SOUTH AFRICA

SUPERVISOR: Prof. John Andrew van der Poll

October 2020

ABSTRACT

There have been numerous studies on formal methods but little utilisation of formal methods in the commercial world. This can be attributed to many factors, such as that few specialists know how to use formal methods. Moreover, the use of mathematical notation leads to the perception that formal methods are difficult. Formal methods can be described as system design methods by which complex computer systems are built using mathematical notation and logic.

Formal methods have been used in the software development world since 1940, that is to say, from the earliest stage of computer development. To date, there has been a slow adoption of formal methods, which are mostly used for mission-critical projects in, for example, the military and the aviation industry. Researchers worldwide are conducting studies on formal methods, but the research mostly deals with path planning and control and not the runtime verification of autonomous systems.

The main focus of this dissertation is the question of how to increase the pace at which formal methods are adopted in the business or commercial world. As part of this dissertation, a framework was developed to facilitate the use of formal methods in the commercial world. The framework mainly focuses on education, support tools, buy-in and remuneration. The framework was validated using a case study to illustrate its practicality. This dissertation also focuses on different types of formal methods and how they are used, as well as the link between formal methods and other software development techniques.

An ERP system specification is presented in both natural language (informal) and formal notation, which demonstrates how a formal specification can be derived from an informal specification using the enhanced established strategy for constructing a Z specification as a guideline. Success stories of companies that are applying formal methods in the commercial world are also presented.

Keywords: commercial software, enterprise resource planning (ERP), first-order logic, formal methods (FMs), formal specification, formal verification, set theory, TLA+, UML, Z, Zermelo-Fraenkel.

MANWELEDZO

Ho no vha na ngudo nnzhi nga ha malugana na Maitele a u khwinisa Sisiṱeme dzine dza konḓa fhedzi hu na u shumiswa huṱuku kha maitele a u khwinisa sisiṱeme dzine dza konḓa kha mbambadzo. Hezwi zwi nga bveledzwa nga zwiṱaluli zwo vhalaho, u fana na vhomakone vha si vhanzhi vha ḓivha maitele a u khwinisa sisiṱeme dzine dza konḓa. U ḓadzisa kha zwenezwo, u shumisa ha dzi zwiga zwa mbalo na zwone zwi siya zwi tshi nga maitele a u khwinisa sisiṱeme dzine dza konḓa a ya konḓa Maitele a u khwinisa sisiṱeme dzine dza konḓa a nga ṱalutshedzwa sa maitele o tou u itelwaho sisiṱeme ane a shumisa zwiga zwa mbalo na kuhumbulele u fhaṱa sisiṱeme ine ya konḓa ya khomphyutha. Maitele a u khwinisa sisiṱeme dzine dza konḓa o shumiswa kha mveledziso ya sofuthuwee u bva 1940, nga tshifhinga tsha ḽiga ḽa u thoma tsha mveledziso ya khomphyutha, U swika zwino, hu na u shumiswa huṱuku ha maitele a u khwinisa sisiṱeme dzine dza konḓa, ane a shumiswa nga maanḓa kha thandela dza sisiṱeme dza ndeme. Vhaṱoḓisisi vhanzhi ḽifhasini ḽoṱhe vha khou ita ngudo dza nga ha maitele a u khwinisa sisiṱeme dzine dza konḓa, fhedzi vhanzhi vha kha u pulana na u langula nḓila hu si khwaṱhisedzo ya kushumele kwa netiweke.

Ṱhoḓisiso iyi yo sedzesa zwihulwane kha uri hu nga engedziwa hani kushumisele kwa maitele a u khwinisa sisiṱeme dzine dza konḓa kha vhubindudzi kana mbambadzo. Sa tshipiḓa tsha ṱhoḓisiso iyi ho bveledzwa furemiweke u leludza u shumiswa ha maitele a u khwinisa sisiṱeme dzine dza konḓa kha mbambadzo. Furemiweke yo sedzesa nga maanḓa kha pfunzo, zwishumiswa zwa u tikedza, thendelano na miholo. Furemiweke i ḓo khwaṱhisedzwa hu tshi shumiwa ngudo u sumbedza khonadzeo yayo. Ṱhoḓisiso iyi i ḓo dovha ya sedza dziṅwe tshaka dzo fhambanaho dza maitele a u khwinisa sisiṱeme dzine dza konḓa na uri dzo shumisiswa hani, na vhuṱumani vhukati ha dziṅwe thekhiniki dza mveledziso ya sofuthuwee na maitele a u khwinisa sisiṱeme dzine dza konḓa.

Sisiṱeme yo tiwaho ya ERP I ḓo kumedzwa nga vhuvhili ha nyambo luambo lwa vhathu (lu si lwa tshiofisi) na zwiga zwa tshiofisi, zwine zwa sumbedzisa uri u tiwa ha tshiofisi hu nga bvisa hani kha u tiwa hune ha si vhe ha tshiofisi hu tshi khou endedzwa nga Tshiṱirathedzhi tsha u Khwinisa tsho Bveledziswaho u fhaṱa Z yo tiwaho. Ṱhoḓisiso iyi i dovha ya sumbedza mvelelo dza vhuḓi dza khamphani dzi no khou shumisa maitele a u khwinisa sisiṱeme dzine dza konḓa.

Maipfi a ndeme: Sofuthuwee ya Mbambadzo, U langula na u ṱanganyisa zwipiḓa zwa ndeme zwa bindu (ERP), Zwiga zwa kuhumbulele zwo khethekanaho, Maitele a u khwinisa Sisiṱeme dzine dza konḓa (FMs), U tiwa ha Tshiofisi, Khwaṱhisedzo ya Tshiofisi, thyeori ya ngudo dza sethe, TLA+, UML, Z, Zermelo-Fraenkel.

OPSOMMING

Vele studies is al oor formele metodes gedoen, maar formele metodes word slegs in ’n beperkte mate in die kommersiële wêreld aangewend. Dít kan aan vele faktore toegeskryf word, soos dat min spesialiste weet hoe om formele metodes te gebruik. Verder lei die gebruik van wiskundige notasie tot die persepsie dat formele metodes moeilik is. Formele metodes kan beskryf word as stelselontwerpmetodes wat die gebruik van wiskundige notasie en logika behels en wat toegepas word om komplekse rekenaarstelsels mee te bou.

Formele metodes word sedert 1940 in die wêreld van programmatuurontwikkeling, met ander woorde, vanaf die vroegste stadium van rekenaarontwikkeling gebruik. Tot op hede was daar ’n geleidelike aanvaarding van formele metodes, wat meestal vir missiekritieke projekte in, byvoorbeeld, die weermag en die lugvaartbedryf gebruik word. Navorsers wêreldwyd doen navorsing oor formele metodes, maar dit handel hoofsaaklik oor roetebeplanning en -beheer en nie die looptydverifikasie van outonome stelsels nie.

Die hooffokus van hierdie verhandeling is die vraag oor hoe die pas waarteen formele metodes in die sake- of kommersiële wêreld aanvaar word, bespoedig kan word. ’n Raamwerk is as deel van die verhandeling ontwikkel ten einde die gebruik van formele metodes in die kommersiële wêreld aan te help. Die raamwerk fokus hoofsaaklik op onderwys, ondersteuningsmiddele, inkoop (*buy-in*) en vergoeding. Die geldigheid van die raamwerk is met behulp van ’n gevallestudie wat die praktiese uitvoerbaarheid daarvan illustreer, bepaal. Die verhandeling fokus ook op verskillende tipes formele metodes en hoe hulle gebruik word, asook die verwantskap tussen formele metodes en ander programmatuurontwikkelings-tegnieke.

’n ERP-stelselspesifikasie word in beide natuurlike (informele) taal en formele notasie aangebied, wat illustreer hoe ’n formele spesifikasie vanuit ’n informele spesifikasie afgelei kan word deur die verbeterde gevestigde strategie vir die opstel van ’n Z-spesifikasie as riglyn te gebruik. Verder word suksesverhale van maatskappye wat formele metodes suksesvol in die kommersiële wêreld aanwend, aangebied.

Sleutelwoorde: eersteorde-logika, formele metodes (FMs), formele spesifikasie, formele verifikasie, kommersiële programmatuur, ondernemingshulpbronbeplanning (ERP), TLA+, UML, versamelingsleer, Z, Zermelo-Fraenkel.

Table of Contents

List of Figures

List of Tables

Publication

The following publication emanated from the research described in this dissertation.

- Nemathaga, A. and Van der Poll, J. A. (2019) 'Adoption of Formal Methods in the Commercial World', in the *Eighth International Conference on Advances in Computing, Communication and Information Technology CCIT*. Birmingham City University, Birmingham, United Kingdom: Institute of Research Engineers and Doctors., pp. 75–84. doi: 10.15224/978-1-63248-169-6-12.

This paper appears in Appendix J.

Acronyms and Pseudo Acronyms

Term	Description
ACP	Algebra of Communicating Processes
ASCII	American Standard Code for Information Interchange
ASM	The Abstract State Machine
BPMN	Business Process Model and Notation
BRS	Business Requirement Specification
CICS	Customer Information Control System
CISR	Centre for Information Systems Research
CSP	Communication Sequential Process
ERP	Enterprise Resource Planning
ES	Established Strategy
FMs	Formal Methods
FRS	Functional Requirements Specification
GUI	Graphical User Interface
HCI	Human-Computer Interaction
HR	Human Resource
IIBA	International Institute of Business Analysis
IT	Information Technology
NL	Natural Language
ROI	Return on Investment
RQs	Research Questions

Term	Description
SDLC	Software Development Life Cycle
UML	Unified Modelling Language
VDM	Vienna Development Method
ZF	Zermelo-Fraenkel

Table 1 Terms, Acronyms and Pseudo Acronyms

DECLARATION

Name: Aifheli Peter Nemathaga

Student number: 51843331

Degree: MSC Computing

Exact wording of the title of the dissertation as appearing on the electronic copy submitted for examination:

Formal Methods Adoption in the Commercial world

I declare that the above dissertation 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.

I further declare that I submitted the dissertation to originality checking software and that it falls within the accepted requirements for originality.

I further declare that I have not previously submitted this work, or part of it, for examination at Unisa for another qualification or at any other higher education institution.

(The dissertation will not be examined unless this statement has been submitted.)

SIGNATURE

20 January 2020

DATE

Acknowledgements

I would like to extend gratitude to my supervisor Prof. John Andrew van der Poll for his assistance, guidance and support throughout my Master's degree.

I would also like to thank the University of South Africa and the School of Computing (SOC) for providing me with an opportunity to pursue my studies further and also offering me financial assistance.

Finally, I would like to thank my son Musiiwa (aged 4), my sisters Mulalo, Adziambei and Humbulani, for always giving me support and pushing me not to give up. I dedicate this work to my late mother, Mrs TC Nemathaga (1950 – 2015).

Dedication

This dissertation is dedicated to my late mother, Mrs T.C. Nemathaga (1950 - 2015), and my four-year-old son, Musiiwa Nemathaga.

Chapter 1 Introduction

1.1 Introduction

This dissertation investigates the feasibility of using formal methods in commercial software development, where in addition to presenting findings, it defines and develops a framework to facilitate the use of FMs in commercial software development. This research focuses on the Enterprise Resource Planning (ERP) system, where a small formal methods specification is written, specifying an ERP system requirements. The adoption of a formal methods framework is validated using a case study to illustrate its practicality.

This chapter gives an introduction as to what formal methods are, and the brief history behind them. The chapter furthermore explains the research focus and also gives the problem statement as to what the research is trying to solve. A list of research questions that will assist in solving the problem is provided. The scope and the research objective is explained. Lastly, the list of the chapters for the rest of the dissertation and the research layout is presented.

1.2 FMs Overview and Context

The advancement of hardware during the past 30 years has led to the creation of large and complex systems. The growing technologies range from mobile devices, industrial machinery and automobiles. These systems require fast processing in order for hardware and software to work together to perform complex tasks (Xilinx, 2012). The lines of codes have increased from 1 to 40 million lines in software and are still increasing. As these systems grow, designers and engineers face many challenges. These systems are designed, enhanced and modified often during their lifetime. Software development is time-consuming and costly, and research has shown that most software does not meet users' needs, and gets delivered out of budget (Bourque and Fairley, 2014). This also applies to ERP systems, that is, ERP project implementation is mostly unsuccessful, or implemented out of timelines, with higher costs (Suryalena, 2013). With such challenges in mind, many software development techniques have been developed to try to overcome them.

Formal methods have shown to be one of the auspicious techniques used to

potentially overcome some of the above challenges. There are numerous benefits of using formal methods. Formal methods have also been shown to reduce the number of defects in software development (Adesina-Ojo, Van Der Poll and Venter, 2011). In the software development world, there is always a search to find better ways of developing software that is free from error and delivered within timelines, on budget. This led to the development of various frameworks and methodologies of developing software. The most famous and widely used is the traditional waterfall methodology, which proposes that software has to be developed using a stepwise approach, i.e. requirements, design, implementation, verification, and maintenance (Royce, 1970). Each stage must be finalised prior to starting the next. The waterfall methodology is one of the oldest models still in use today (Palmquist et al., 2013). Yet, many of the waterfall projects are delivered out of budget, with many defects and the end-product usually does not present the real needs of the user (Pressman, 2009). Furthermore, ERP system project implementation failure can also be attributed from two aspects; these are organisational aspects and technological aspects.

There is an increased uptake of the Agile methodology in the commercial world, where software is developed incrementally, and in rapid cycles. Agile's main objective is to deliver value to the customer by means of working software (Beck et al., 2001; Palmquist et al., 2013). Agile is guided by a manifesto stating the principles that ought to be followed when using Agile. That said, Agile has many disadvantages, such as a lack of documentation, and the project may easily go off track if the customer's requirements are not understood. The aforementioned formal methods can be incorporated into any stage or phase of the Software Development Life Cycle (SDLC) and have proven to reduce the count of errors (George & Vaughn, 2003; Srihasha & Reddy 2015).

Software testing has traditionally been the only technique that has been used and is still been used to find defects. Testing code is not an effective way of finding subtle bugs/error in design. The use of formal methods helps to reduce errors early on in software development, thereby saving on the cost of software projects. Formal methods are categorised in two main groups: i.e. 1) pure mathematics, this is challenging and is mostly not used in the real world; and 2) software engineering, which focuses on creating increasingly better software (Kneuper, 1997; Van der Poll, 2010).

Formal methods use discrete mathematics and logic to verify and analyse models at

any stage of the development process (Woodcock et al., 2009). The most significant part of the development process is to understand the needs of the user. Furthermore, according to George and Vaughn (2003), formal methods are useful when gathering, articulating, and representing requirements. This, then, assists the programmer in developing a system that meets the user's needs.

A formal specification can be written in a state-based technique, which involves the creation of state machine specifications, simulation proofs, and abstract functions. During the implementation level, formal methods are used to verify code by attempting to prove theorems (proof obligations) about the implementation. Some tools used for formal methods can automatically generate compilable code e.g., B-method (Ilić, 2007). The clarity, completeness and consistency of a formal specification facilitate the derivation of test cases (Tretmans and Belinfante, 1999). As part of this research, a formal specification will be documented using the Z notation, which is a formal specification language.

1.3 Research Focus

The core emphasis of this study is on the adoption of formal methods in business or the commercial world, placing more emphasis on the ERP system. This research will investigate, through the use of document analysis, the companies that are using formal methods and the benefits they realised from formal methods. The recurring failures of the commercial systems will also be investigated. In addition, the research will also analyse the scholarly literature on these aspects and subsequently, shall look at ways to facilitate the adoption of formal methods in the commercial world.

1.4 Problem Statement

The use of traditional software development processes is widespread in the commercial world. The most common is the waterfall model where software development is done in sequences, namely:

1) Requirements elicitation;
2) System design;
3) Implementation (coding);
4) Verification (testing); and
5) Post-delivery maintenance.

As mentioned above, a new technique that is starting to gain fame in the software development industry is the Agile method. Agile aims to continuously deliver value to the customer. Despite these advances, the software is still delivered late, out of budget, and with a considerable number of defects (Bourque and Fairley, 2014).

Using FMs in software development stages can arguably yield many benefits when it comes to software quality. One of the key benefits of utilising FMs is that it alleviates the problem of ambiguity, where formal methods give a full understanding of requirements and software design (George and Vaughn, 2003; Gilliam, Powell and Bishop, 2005). This leads to the reduction of defects in requirements and design and testing becomes easier. That said, there remain several challenges with using formal methods, such as expense, time, and the extensive training required, as few developers and engineers know how to use it (Spichkova, 2012a).

Given the benefits and advantages of FMs, there appears to be a slight commercialisation of FMs, but the use of FMs remains mostly in universities and mission-critical projects (Di Vito, 2014). Hence, the problem addressed in this research is the slow adoption of formal methods in the commercial world.

1.5 Research Questions

From the above problem statement, we formulate the following research questions (RQs):

1. What makes Formal Methods projects successful?
   1.1. To what extent can FMs improve on the quality of ERP development?
2. Why is there a slow adoption of formal method in the commercial world/Business?
   2.1. What is the status quo of the use of FMs in the commercial world/Business?
3. What can be done to increase the adoption of Formal Methods in the commercial world/Business?

1.6 The Scope

The field of formal methods is broad, with numerous challenges that still require further research and clarification. Formal methods can be useful in the SDLC, where, as interest in the use of formal methods continues to grow, a considerable number of researches (Woodcock et al., 2009) are been carried out on each type of

formal method.

This research will mostly be theoretical in nature, focusing on the formal specification phase of formal methods usage, consequently, the scope of the research includes:

- the utilisation of FMs in business;
- formal methods specification;
- reason for slow adoption;
- myths around formal methods;
- ERP System formalisation; and
- a mechanism to facilitate the adoption of FMs in the commercial software world.

1.7 Delineations and limitations

The following lie outside of the scope of this research:

No code will be generated as part of this research. With no code there will be no working software and testing will not be conducted. As indicated above, this research is mainly theoretical in nature. No prototype of the ERP system will be produced based on Formal Methods specification outlined in the coming chapters.

1.8 Research Objectives

Since this research aims to investigate the reasons for the slow adoption of formal methods in business, followed by recommending measures to alleviate such challenge, our objectives are to:

1) determine the failures of current commercial software development;
2) assess literature pertaining to formal methods to determine what makes FMs projects successful;
3) determine the status quo of the use of FMs in the commercial world/Business through literature review;
4) determine the reasons for the slow adoption of FMs in the commercial world/Business; and
5) develop a framework to facilitate the adoption of FMs in the commercial world/Business.
    - Validate the framework using a case study.

1.9 Dissertation Layout

The following section will present the dissertation layout by explaining the number of chapters and the summary discussion of what each chapter entails.

1.9.1 The list of chapters

The following section will discuss the chapters contained in this dissertation:

Chapter 2 Literature Survey

Chapter 2 details the ERP system, in terms of what an ERP system is, and what modules are an ERP system comprise of. In addition, the challenges that arise when implementing an ERP system within the organisation are discussed. Chapter 2 then touches on formal methods putting more emphasis on the Z specification language. Furthermore, the chapter will explain the differences between informal and formal methods in tabular form. Types of formal languages are identified, and the minimal description of Z notion is conversed. Myths around formal methods are identified. Lastly, Chapter 2 will focus on the reasons why there is slow adoption, suggesting the ways to hasten the adoption of formal methods in the commercial world. To close off the chapter, practical examples of the use of formal methods in the commercial world are discussed.

Chapter 3 Informal, Semi-Formal and Formal Specification

Chapter 3 documents ERP specification in an informal way/natural language, semi-formal and the formal way. This is structured by providing a case study first, followed by the specification. Parts of the informal specification are discussed i.e., a UML process diagram, use case diagram, and the details of the process and use case diagram in a tabular format. Before writing a formal specification, a brief introduction to mathematical set theory is presented. The last section of this chapter is the formal specification using the Z notation on the purchasing module of the ERP. To close off the chapter, a preliminary framework is presented.

Chapter 4 Research Methodology

This chapter explains the research philosophies used for this paper and the reason why a certain philosophy is followed. This is described using Saunders et al.s (2015) Research Onion. Each step of the diagram is explained, and a reason is provided if relevant to the research or not. Furthermore, this paper expands deeper into the research methodology incorporated, as well as data collection methods used.

Chapter 5 Adoption Framework

Chapter 5 presents a framework regarding how to accelerate the adoption of formal methods. The conceptual framework is named the Formal Methods Adoption Framework. Each element of the adoption framework is discussed in a tabular format and lastly, a framework diagram will be presented. The framework is linked to the propositions presented throughout the dissertation.

Chapter 6 Framework Validation

This chapter validates the framework otherwise putting the framework in practice. The validation is in the form of a case study. From the case study, an explanation of how each step of the conceptual framework will be implemented is presented.

Chapter 7 Conclusion and Future Work

This chapter completes the research. It achieves this by giving the summary of the findings and how they relate or answers the research questions. It further explains the shortcomings of this research. Lastly, it details the future contributions still required to be done on this topic.

1.9.2 The relationship between chapters

The following diagram illustrates the research layout, i.e. how the research is structured. The diagram also shows the main parts of each chapter.

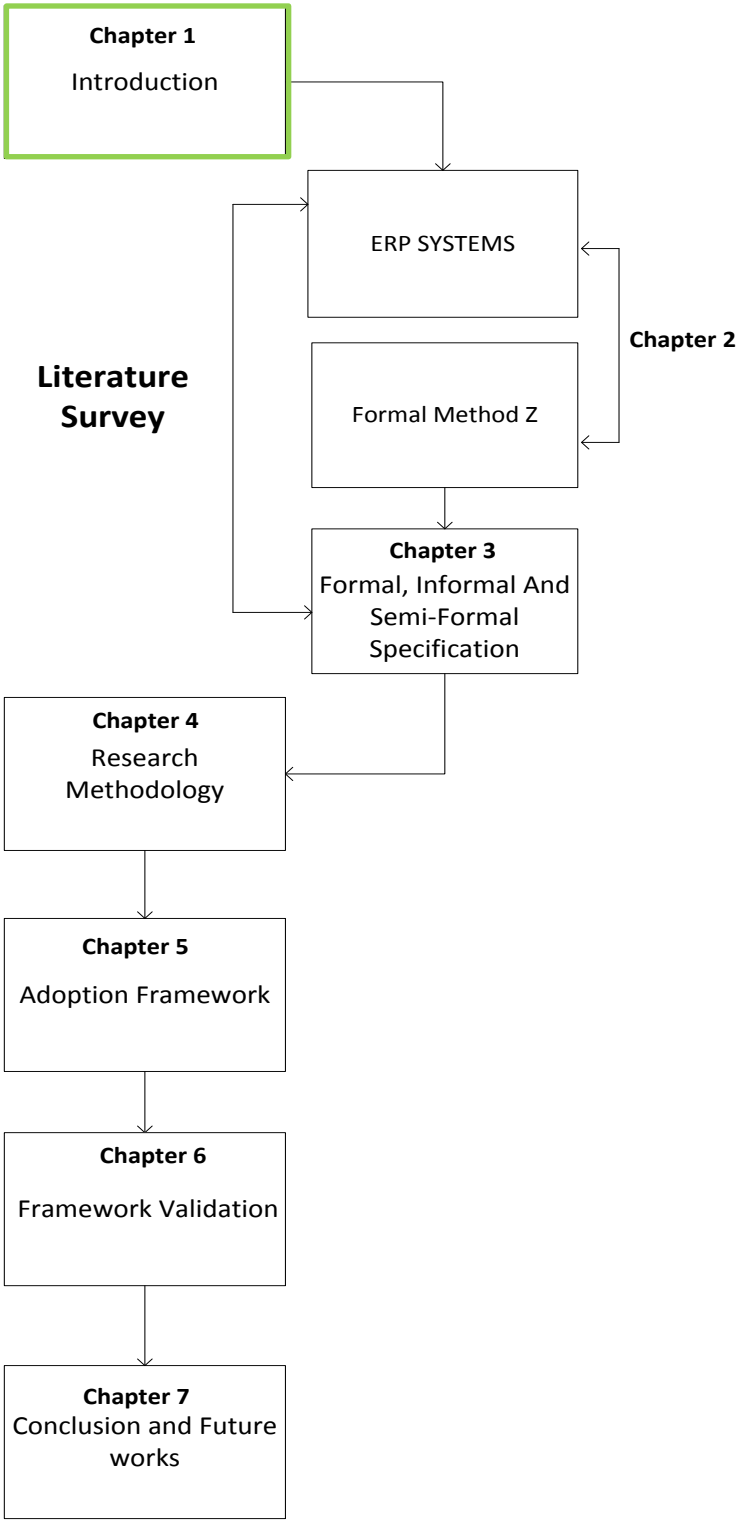


Figure 1-1 Dissertation Layout

1.10 Summary

This first chapter's goal was to set the scene for the dissertation. It gave an overview of what this research is aiming to archive. This chapter also gave an introduction of what formal methods are, and the brief history behind them. The research scope and problem statement, that is, why formal methods in the commercial world are infrequently used. The research questions to help solve this problem were listed. Furthermore, the research objective and the dissertation layout were presented.

The next chapter will discuss aspects around ERP systems, in terms of what an ERP system is, and what modules an ERP system comprises. In addition, the challenges that arise when implementing an ERP system within the organisation are discussed. Chapter 2 then expands on formal methods, putting emphasis on Z-specification language. In conclusion Chapter 2 will address the reasons why there is slow adoption and suggest ways to fasten the adoption of formal methods in the commercial world.

Chapter 2 Literature Survey

2.1 Chapter Layout

The below diagram shows where we are in this dissertation, the green boxes highlight the sections of Chapter 2, i.e. ERP Systems and the Formal Method Z.

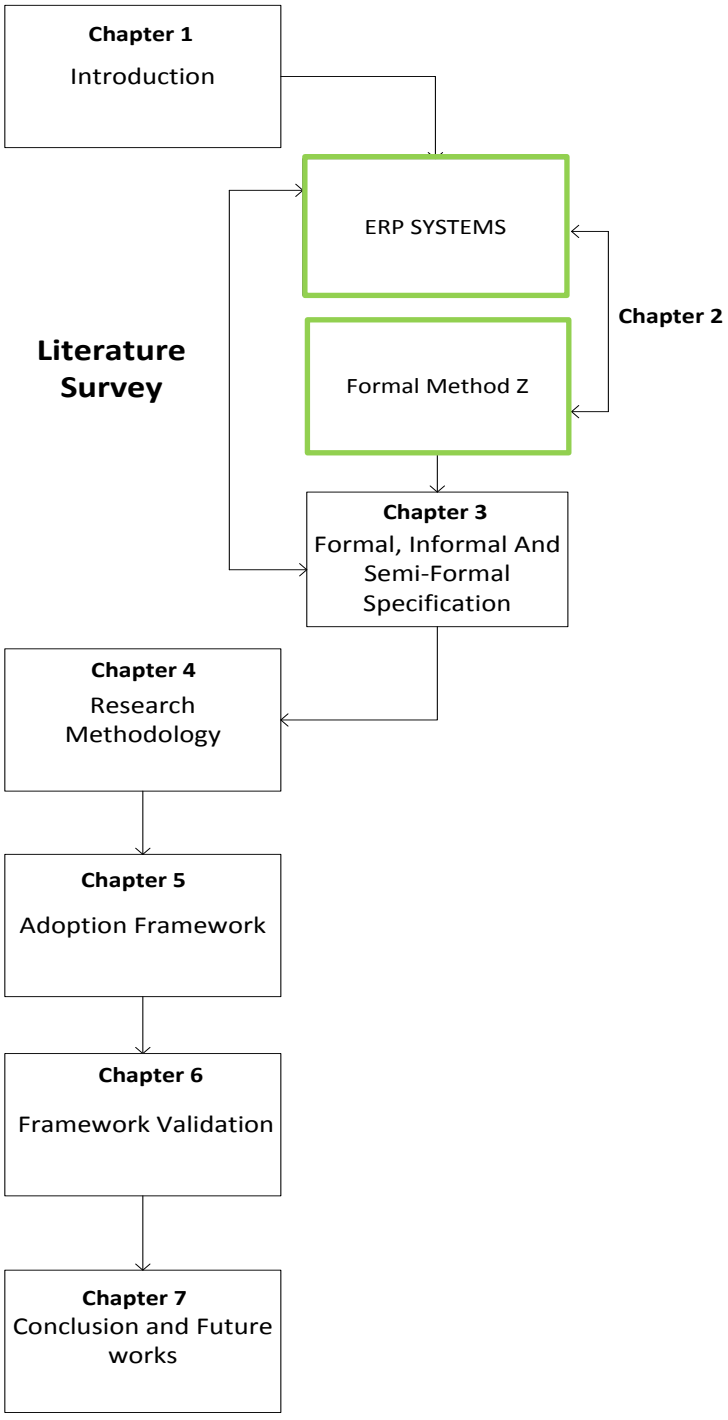


Figure 2-1 Dissertation Layout

2.2 Introduction

The previous chapter presented an overview of what formal methods are and the history behind FMs. Furthermore, Chapter One explains the focus of the research, in terms of the problem that the dissertation is trying to solve. The research questions that will assist in solving the problem are listed, as well as the scope and the objectives for the research. Lastly, a research layout and a brief description of each chapter is presented.

This chapter will discuss the ERP system, in terms of what is an ERP system and what modules an ERP system comprises. In addition, the challenges that arise when implementing an ERP system within the organisation are discussed. This chapter then touches on formal methods, placing more emphasis on the Z-specification language. Furthermore, the chapter explains the differences between informal and formal methods in tabular form. Types of formal language are identified, and the minimal description of Z notion is discussed. Myths around formal methods are identified. Lastly, the chapter focuses on the reasons why there is slow adoption and suggests the ways to fasten the adoption of formal methods in the commercial world. To close off the chapter, practical examples where formal methods are used in the commercial world receives a discussion.

2.3 What is an Enterprise Resource Planning (ERP) System?

This research will place focus on formal methods for the specific system which is ERP. This part of this chapter will explain what the ERP system is, and the next chapter will focus on writing a formal specification for an ERP System.

Enterprise Resource Planning (ERP) software is defined as a combined software programmes clustered into standard functional modules i.e., Procurement, human resources, finance, contract management etc. developed by a vendor or in-house (Shehab et al., 2012), involving “One database, one application and a unified interface across the entire enterprise” (Babu and Bezawada, 2012). Some ERP systems can be purchased off the shelf, then customised to meet specific customer needs. ERP systems assist businesses in performing their daily operations, which can bring massive benefits to the organisation. But, with these benefits, ERP project implementation is mostly unsuccessful, or implemented out of timelines, and with

higher costs (Suryalena, 2013).

Failure of ERP project implementation can be attributed to different factors, such as unclear requirements, project managers focusing on the financial aspect of the business and neglecting other parts of the project, and lack of proper software development processes in place to manage the projects, to name a few. Most of the time, the success of the project is attributed to delivering the project on time and within budget, where the tendency exists to forget the users of the system and the smooth transition from the previous process to the new one (Markus, Tanis and Van Fenema, 2000). The use of formal methods will help alleviate most of the problems when implementing an ERP system within the organisation.

The successful implementation of ERP systems can be grouped into two aspects, that is, the organisation and the Technological part. According to Sangster et al. (2016a) organisation aspects can be:

- effective organisational change;
- user involvement and participation on the project; and
- trust between partners or stakeholders.

Examples of technological aspects:

- an acceptable implementations strategy;
- avoid too much customisation; and
- the right version of the ERP system and the correct knowledge of the legacy system.

The next section will present an ERP architectural diagram showing different modules linking to one database.

2.4 ERP Modules

The following ERP architecture diagram presents some of the main modules that are contained in the ERP system:

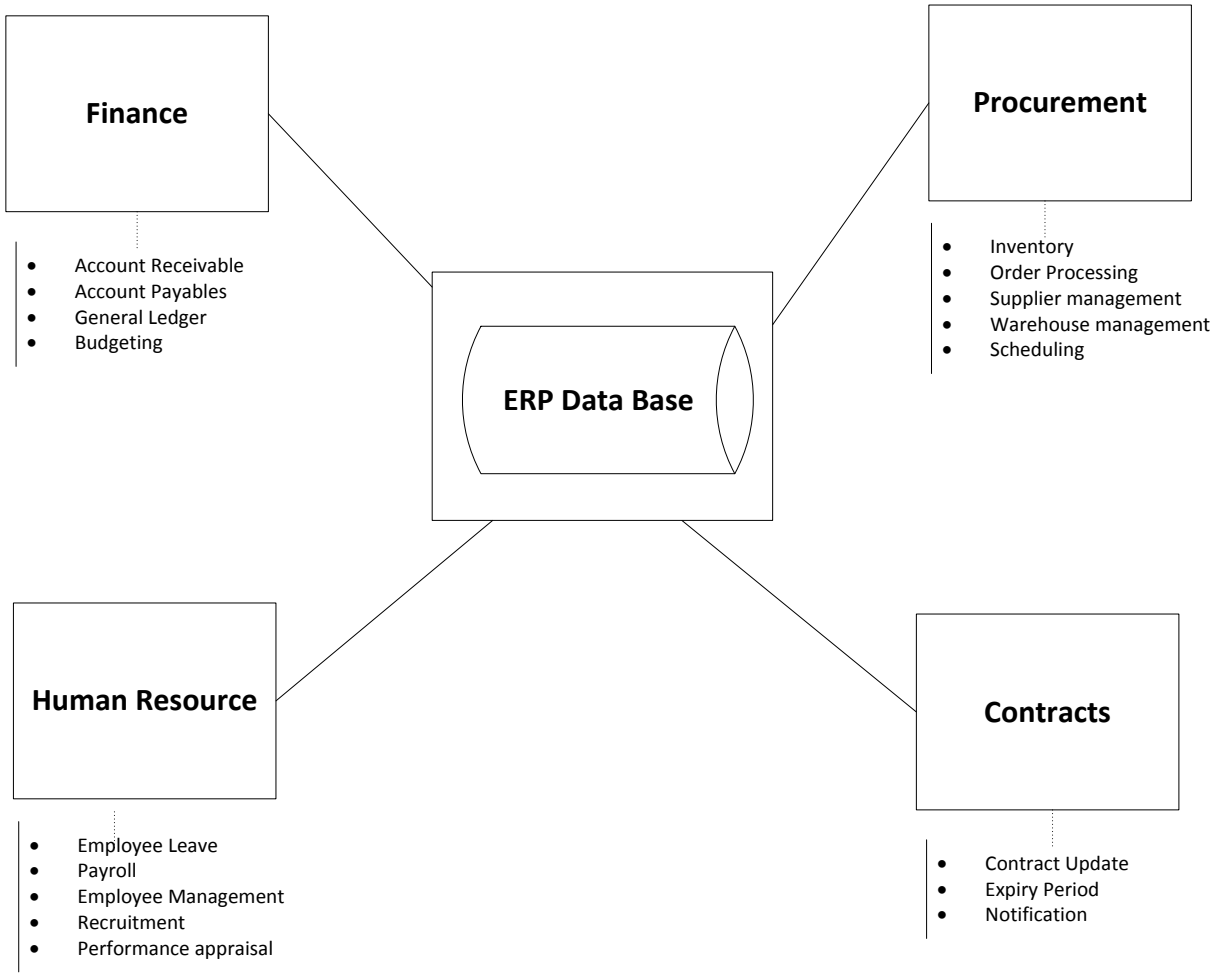


Figure 2-2 ERP Architecture (Kilic, Zaim and Delen, 2015)

Below we present a brief description of ERP modules and how they benefit the organisation. The main idea behind ERP is to provide the right information to the right people at the right time. This improves organisation performance significantly. Other important aspects of ERP include that it is linked to the organisational strategy, the organisation structure, processes, and IT systems (Subramoniam et al., 2009). The first module to be discussed will be the financial module.

Finance is critical to the organisation, and it impacts almost every part of it. This can be from sales to procurement and human resources. The information produced by the financial module helps the decision-makers to formulate strategies to gain a competitive advantage over other organisations. The most common functionalities

of the finance modules are financial accounting (GL- General Ledger, Accounts receivables and payables); investment management (budgeting, controlling); and treasury.

Procurement as a module deals with the purchasing of materials for internal use or resale within the organisation. Procurement mostly involves a workflow build to automatically evaluate a supplier and measure the inventory at hand. Lastly, most purchasing modules are integrated to invoice verification. Gao, Zhang and Wang (2008) call the procurement module the internet procurement.

Human resource (HR) This module is used by the human resource department to manage human resources or employees within the organisation. Part of the function of the HR module is to manage employee information, such as names, contact details, and location. HR module is also used for the recruitment process of the new employee. The payroll system also resides under the HR module, which assists in managing employee salaries and payslips. Employees can also use the HR module for leave applications and to perform performance reviews alongside with their managers. Some HR modules have time & attendance component embedded in them (Cardoso, Bostrom and Sheth, 2004).

The Contract Management module is used to manage contracts within the organisation. This can refer to suppliers/vendor contracts, or clients' contracts. This module has information about the contract, such as the start and expiry date of the contract. The contracts are linked to the materials or services that the supplier sells to the organisation. In a large organisation, this module benefits a great deal, as the organisation can have thousands of contracts with different suppliers which becomes a nightmare to manage manually.

Implementing ERP modules within the organisation can be challenging, where many aspects of the project can go wrong. When using traditional methods, such as waterfall, the cost of the project can increase significantly. From the requirements stage to the implementation stage, the costs of fixing errors/defects rise. Today's ERP systems are mostly web-based, meaning that they can be easily accessible from different devices and in different locations (Subramoniam et al., 2009). In around the year 2000, the Gartner group presented a new terminology ERP II to name the latest upgrades in the ERP systems. ERP II is otherwise known as the next generation ERP (Subramoniam et al., 2009). The key modification from ERP to ERPII is that the latter is more web-friendly, and it allows for a wider integration between department and industries (Felderer et al., 2016).

2.5 Challenges of implementing and using an ERP system

Large organisation system integration can be difficult or incompatible, and this will also require the process of re-engineering and change to organisational culture (Bernroider, Wong and Lai, 2014). The impact on the small and medium enterprise may be minimal, as they are more flexible, and a change management process may run smoothly.

Off-the-shelf ERP systems offer generic requirements, and this leads to more customisation to meet organisation-specific requirements. Customisation is costly, time-consuming, and can become very challenging when implementing ERP upgrades (Kwahk and Ahn, 2010).

In terms of the costs, the EPR system requires a high upfront investment fee, and it proves challenging to recognise the ROI (Return On Investment), as it is a long-term undertaking. Furthermore, the maintenance and user support fees are very high, which leads most companies to opt-out of having an ERP system (Elbertsen and Reekum, 2008).

However, the benefits of having a working ERP system implemented within the organisation outweigh the challenges of implementing ERP systems (Equey et al., 2008).

According to Pang (2016), from Gartner, the top 5 most used off-the-shelf ERP systems are:

- SAP
- Microsoft Dynamics AX
- Sage X3
- Infor
- Oracle

Gartner further provides an ERP Quadrant showing the most-used ERP system in terms of market share and revenue:

Figure 2-1 Gartner ERP Quadrant (Softwareshortlist, 2015)

Asgar and King (2016) propose a bipartite graph approach, which is a lightweight formalisation to map the requirements of legacy and new off-the-shelf ERP system. In a traditional software implementation process, ERP implementation is comprised of the following stages: FGA (Fit Gap Analysis), which involves ascertaining customisation requirement and business process requirements; thereafter, design and development; which follows data migration from the old system to the new system; then, testing, and lastly user training and deploying the system to live environment (Asgar and King, 2016). In an Agile methodology, these steps occur incrementally, and in an iterative manner.

Incorporating formal methods during requirements specification, analysis and design contribute significantly towards the success of ERP implementation. This allows for the early detection of errors during the documentation stage.

The aforementioned presents what ERP systems are, the following section will discuss formal methods, and the next chapter will formularise ERP requirements.

2.6 What are Formal Methods?

The push to use formal methods in business has been the main focus of researchers and practitioners for some time now. Even with the benefits of a reduction in defective software and production of systems within timelines, formal methods adoption by the business world is slow (Iddiqui, Akhter and Ian, 2014).

Formal methods are defined as a system design method that uses mathematical notation and logic to build computer systems (Bourque and Fairley, 2014). According to Lockhart, Purdy and Wilsey (2014), the use of mathematical-based modelling makes system behaviour more logical. Formal methods can be useful in the development process when verifying and clarifying the requirements (Crepaldi, 2005). Formal methods assist in clarifying customer requirements, removing ambiguity, incompleteness and inconsistency, and lastly facilitating the communication of requirements and design. According to Van der Poll (2010), a formal requirement specification may be amenable to programmed analysis and reasoning.

The following diagram shows the traditional methods of the software development process. The most famous and widely used module is the traditional waterfall model. This model was presented by Royce (1970). This is a document-driven approach, where when each phase is completed, a document needs to be produced. This is otherwise known as a plan-focused process, where in practice, you plan and schedule all the process actions before work can begin on them.

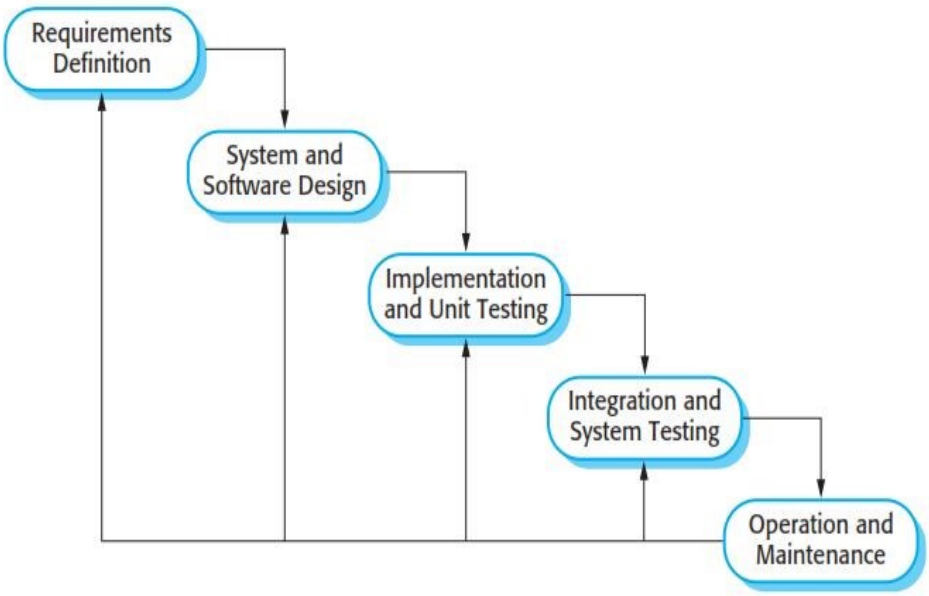


Figure 2-2 Waterfall Model (Crepaldi, 2005)

The stages of the waterfall model are:

1. Requirement analysis and definitions: at this stage requirements are elicited and defined by consulting affect stakeholders or system users. This can be done using various techniques, such as interviews, focus groups, documents analysis, prototyping, or observations. These requirements are written and agreed to on the requirement specification document. Requirements should have the following "SMART" characteristics: Specific, Measurable, Attainable, Realisable, and Traceable (Nathan and Scobell, 2012).

2. System and software design: at this stage a technical document or system design document is produced. The document allocates requirements written in stage one to the hardware or the system and also creates a system architecture. Formal methods can be very useful at the stage.

3. Implementation and unit testing: at this stage, the actual coding using various languages such as C#, C, Java starts, and the testing proceeds. After each module or unit is produced, the unit gets tested. This can either be by testers, or the coders themselves. Most of the errors can be injected at this stage as a result of incorrect requirements and the design document (Schach, 2011).

4. Integration and system testing: all the developed software units are integrated or put together to formulate the final system. After integration, the software is tested to make sure that everything works. Different types of test are conducted such as system testing, regression testing, and UAT with the owner of the system or the software (Crepaldi, 2005).

5. Operation and maintenance: the system is delivered to the owner and then maintained. Maintenance includes correcting errors not previously discovered, as well as doing system upgrades. Change requests also form part of maintenance. The system can be maintained over a period of time, as per the agreement after which it can be decommissioned or retired (Suryn, 2014).

At each stage, a document is produced and approved (signed-off). In reality, all these steps overlap with one another and there are minimal iterations within the stages. Sommerville (2005) suggest this model ought to be used when requirements are well understood, and they might not change drastically.

The amended version of the waterfall model involves formal system development, which is part of formal methods. The system specification is developed using mathematical models, the mathematical model can be transformed into executable code (Crepaldi, 2005).

The following graph (diagram) shows the numbers of bugs or errors that are introduced/ inserted during each stage of software development, where the graph further illustrates the costs of fixing these errors in each stage.

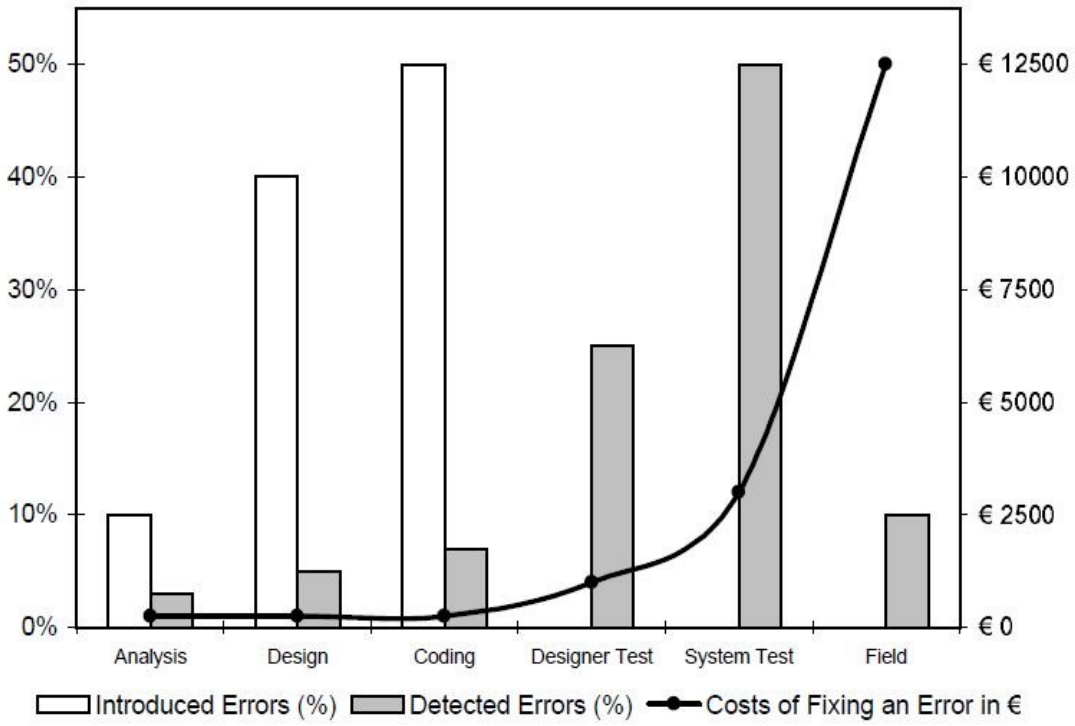


Figure 2-3 Introduction, detection and costs of errors in the design trajectory (Atlee et al., 2013)

The graph reveals that fewer errors are detected from the analysis phase to the coding phase, where more errors appear as the result of the incorrect analysis, or not understanding the requirements correctly.

The cost of correcting the errors increases with each phase, mostly during system testing and designer test phase. It is easier to correct the requirements on the document than to fix a system that is already implemented. Formal methods emphasise that more time ought to be spent on the requirements phase or the analysis phase by developing the formal specification, which will help minimise the costs of fixing errors at a later stage.

2.7 Common types of software failures

Process failure is caused by poor project management during the software development process, a lack of communication, and the choice of the software development methodology that does not suit the project. An example of this type of failure is the crash of Korean Airlines Flight 801 into a hillside, resulting in 228 fatalities, due to the modification of the flight system, which could not calculate the

required radius of 55 nautical miles or 102 kilometres. This can be attributed to negligence (Ogheneovo, 2014).

Real-Time Anomalies: refers to a software bug. An example of this is a Russian spacecraft that had a software error, which caused it to land 300 miles of the target also causing causalities.

Accuracy: this applies mainly in calculations, loss of accuracy when converting an integer to float, or the division of two integers, which ought to result in an integer, not a decimal. This can results in an error if error handling is not done properly. An example is a Patriot Missile that experienced a software precision error causing it to miss its target.

Abstraction: refers to a lack of data abstraction, which causes defects in a given code or system. Abstraction mechanisms are required so as to ensure the proper running of the system (Charette, 2005). An example of this is the software incorrectly reading the year between 99 and 00, where software algorithm interprets the year 2000 as the year 1900.

Constraint: an example of constraint failure is a buffer overflow and stray pointer. New languages such as C# and Java have a mechanism to do constraint lookup on data types, which helps with data type violation. Another method is the use of Sandbox (Ogheneovo, 2014).

Reuse: this involves using existing software components to develop new software. (Crepaldi, 2005). This is done in order to improve the maintainability and quality of the system, and also to reduce development timelines and the costs. When reusing artefacts that already have defects, the defects automatically filter down to the new system. Proper testing is required before reusing artefacts.

Logic: These are flaws in logic processing or incorrect workflows. An example is an upgrade in the AT&T system of 1990. The upgrade caused a switch to result in errors that caused the routing of traffic to other switches. The switch was sending “out of service” message, which caused other switches to crash. Upon investigation, it was discovered that the failure was as a result of a missing break statement in code. 60 million in revenue was lost was a result of this.

Faulty code: this refers to a code that is poorly written, which can be easily hacked

and which may cause runtime errors.

Operator errors: as caused by the users of the software. This can be due to user not knowing how to use the system, or insufficient training as to how to operate the system (Schach, 2011). This results in the user injecting errors into the system.

Boca, Siddiqi and Bowen (2010) categorise the general type of faults encountered on the systems in the following ways:

- the delivered system does not meet the customer's requirements;
- incorrect design i.e., by the solution architecture; or
- faults in implementation or coding.

Further examples

In mission-critical systems, early detection of errors and subsequent correction of such errors are paramount. Intel® lost almost $500 million while trying to fix error on their Pentium chip which was already produced in mass numbers (Kaivola, 2011). Five billion dollars was used to fix a problem with a flight system in June 1996 in the Ariane 5 rocket Ariane where, about 40 seconds after take-off, the rocket launcher shut down, and lost control due to system failure, causing buffer overflow (Lions, 1996; Crepaldi, 2005).

2.8 Differences between formal and informal (natural language) specifications

Informal (Natural Language Processing)	**Formal Methods**
Each stakeholder has its own interpretation of the requirements	A complete and comprehensive view of system requirements
More errors and if not correct can result in high project costs	Less error and blunders in the document
Uses a combination of graphics and semiformal notations	Uses mathematical notation
No need for mathematics just knowledge on the software engineering domain	They need someone to be mathematically literate
They leave space for inconsistency and ambiguity	Provides conciseness, clarity and unambiguity

They ideal for eliciting requirements	Allows the engineer to produce high-quality systems

Table 2-1 Differences between formal and informal specifications (Ilić, 2007)

Both formal notation and natural language processing (NLP) can result in a vague understanding of the system (Li et al., 2015). All this depends on the engineer or the developer understanding what to build, irrespective of the language used on the specification. It is possible to learn the formal language, but it takes time, and is also costly, being dependent on the user's willingness to learn the language.

Figure 2-4 shows the types of formal methods. They are grouped into two, viz. algebraic- and model-based specification styles.

	Sequential	**Concurrent**
Algebric	Larch (Guttag, et al., 1993) OBJ (Futatsugi, el al., 1985)	Lotos (Bolognesi and Brinksma, 1987),
Model-based	Z (Spivey, 1992) VDM (Jones, 1980) B (Wordsworth, 1996)	CSP (Hoere, 1985) Petri Nets (Peterson, 1981)

Figure 2-4 Formal specification languages (Crepaldi, 2005)

Figure 2-4 also shows the inventor of a certain formal language. From the above figure, we observe that most of the languages were developed in the 1980s.

The two main types of formal specification techniques are algebraic (also called property-oriented) and model-oriented.

Property-oriented: algebraic i.e., established on equational axioms, or axiomatic founded on first-order logic used to specify system properties in a declarative-methods style.

Model-oriented or model-based: firstly, a system abstract model is specified where, on the abstract model, "states" are created which are the static properties of the system using mathematical set theory. Next, first-order logic is used to construct operations on those states. From Figure 2-4, examples of model-based languages are Z, VDM, and B.

The two techniques do share a common aspect, namely, they both use mathematical notations, which is first-order predicate logic used to define how the system ought to behave and also to share static properties with each other (Crepaldi, 2005).

Below are examples of model-based techniques, the most prominent being the Z specification language.

- Abstract State Machines – The Abstract State Machine (ASM) proposition implies that any algorithm can be modelled by an appropriate ASM (Börger Egon and Stärk Robert, 2003). ASM bridges the gap between the two ends of system development, viz. human understanding, and the formulation of real-world problems, by deploying an algorithmic solution through executing-code machines on changing platform. When compared to UML, ASM claims to have a simple scientific background, which adds more precision to the realism of the method.

- B-Method – B is a formal method for the development of programme code from a specification in the Abstract Machine Notation (Cansell and Méry, 2003). B can be considered to be a formal method that covers the entire SDLC from requirements, system design, implementation, and post-delivery maintenance. B can be written using the B-Tool interpreter, which helps with identifying syntax errors. B has been used in many mission-critical systems, such as train control systems, and smart cards.

- Z – A specification language used for describing computer-based systems; based on set theory, and first-order predicate logic (Banerjee, Sarkar and Debnath, 2016). B – Method is more similar to Z, as it was developed after Z, and stems from Z. It will be discussed in detail in the next section of this chapter.

Process-based – the most commonly used and successful process-based formal method language is CSP and ACP. This type of formal methods allows engineers to specify systems that are running concurrently, and at the same time integrated to one other, by sharing information (Hoare, 2015). ACP and CSP use an axiomatic algebra approach to give a formal definition to various operators of the system. ACP essentially uses an axiomatic, algebraic method to the formal classification of its several operators.

Axiomatic – Axiomatic systems can be used together with logically derived theorems. Mathematical set theory has been around for a very long time. Enderton (1977) indicates George Cantor as the father of set theory. Gottlob Frege further published a book around 1893 and 1903 demonstrating how mathematics can be developed from the philosophies of set theory. ZFC is a formalisation of set theory.

The formal methods categories summarised above, and the type of languages associated are not exhaustive, but this research employs the model-based language Z.

In addition, there are semi-formal specification languages. A widely used semi-formal specification language amenable to formalisation is UML (Unified Modelling Language). Ma (2008) proposed that more focus ought to be directed to class constructs by considering case studies by means of which to achieve the formalisation of UML. There has been a lot of work on formalising UML using variants of description logics.

The practice of FMs is made up of a number of components and activities i.e., formal specification, formal proofs, model checking, and abstraction. The construction of a formal specification involves translating natural language, Diagrams, tables etc. to a mathematical specification, and this includes a description of high-level behaviour and the properties of the system. Formal specifications have various types or forms, such as a model-oriented system, which refers to the construction of system behaviour using models i.e. state charts, sets etc.

The next activity in the use of FMs is conducting formal proofs, considered to be one of the most essential parts of a formal specification. Formal proofs are constructed as a sequence of small steps, each of which is justified using a small set of inference rules. Proofs can either be done manually or automated (Schneider, 2004).

Some formal methods involve model checking, which is a technique based on constructing a fixed model of software and verifying that the desired property speaks to that model. The main disadvantage of model checking is that it involves many processes. Baier and Katoen have noted that “any verification using model-based techniques is only as good as the model of the system.” (Laroussinie, 2010,

p.8). Fisher (2011) and Schneider (2004) both agree that formal verification provides a way of possibly knowing the perfection of a system in all possible conditions. Formal verification also offers an alternate to assure that the software is fully free of errors e.g the use of Armada tool for verification (Lorch et al., 2020). In summary, model checking refers to using some software to automatically check that the software satisfies its specification.

Lastly, we have abstraction, which involves the use of smaller models to represent a programme. When constructing a specification, obtaining the correct level of abstraction is very significant. Using smaller models allows the designer to focus on the most important characteristics and fundamental properties (Fisher, 2011).

Schneider (2004) indicates that more time is spent on design simulation, where the defects found in the later stages of the design results in a high cost of the redesign, leading to delays in marketing time. The idea with formal methods is to spend more time in the specification phase to get it correct, thereby leading to the reduction of time spent on the design and the actual coding. Given the aforementioned, the final product ought to be correct.

Z specification language will constitute a fundamental part of this research and is discussed next.

2.9 The Z Specification Language

The Z language was established in late 1970 at Oxford University by the Programming Research Group, otherwise known as the PRG. Banerjee, Sarkar, and Debnath (2016, p4.) write that the “Z-notation based on the formal specification of a component model has been proposed to develop a component model formally”. Z is based on first-order logic and a strongly-typed fragment of Zermelo-Fraenkel set theory, and embodies numerous rich notations. Using a formal specification language such as Z, software systems can be designed with minor uncertainties (Hussain, Dunne and Rasool, 2013). Type checkers and Latex style files for writing Z notations have been developed, as Z is written mostly in non-ASCII mathematical symbols.

A Z specification comprises of schemas and is accompanied by narrative text. A schema is an organising unit to hold logically associated mathematical notation. Formal methods comprise of the following logical operators:

- ¬ negation
- ∧ conjunction
- ∨ disjunction
- ⇒ implication (note: not →)
- ⇔ equivalence (note: not ↔)

Schema Example

Figure 2-5 Schema

The schema usually divided into two parts i.e.:

- Part One: where the variables (components/declarations) and the types are presented; and
- Part two: predicating constraints assigned to the values of the variables/components.

The following are advantages of Z as a choice of a formal specification language: (Hussain, Dunne and Rasool, 2013)

- the use of Schemas makes Z easy to read;
- a well-written Z specification can be used as a manual for the system;
- the flexibility to model a specification can lead directly to the code;
- a large class of structural models can be described in Z without higher-order features, and can, therefore, be analysed efficiently; and
- independent (e.g. error) conditions can be added later.

Spivey (2010) added that using formal notation helps in understanding how the system will operate, and it allows the designer more choices about the design of the system. The omitted parts of the specification become easy to identify, and the overall document quality is increased.

Z has some disadvantages, however (Adesina-Ojo, 2011; Dongmo, 2016):

- for complex software that generates a big specification, it may be hard to produce a number of state and operation schemas;
- Z fails to provide for grouping of operations on a particular state;

- some classes of the system are still difficult to specify;
- there are not many industrial tools that can be used to write and verify a specification;
- it becomes difficult to manage and group schema structures for large systems when using Z; and
- Z does not clearly handle exception handling.

If only mathematics is only used, the formal specification will become hard to manage and difficult to read it, therefore Z specification is written in conjunction with natural prose. For example, when formal methods are removed from the specification, the specification ought to remain readable and understandable. Z specification describes the “what” meaning, that is, what the system does, and not how it does it. The final design of the specification can be executable by the computers in some instances, it is also designed to be readable and understandable by the humans. Z is also guided by renowned pseudo-algorithm called Established Strategy, which assists when writing a Z specification (Van der Poll and Kotzé, 2005)

Z specification can be written or produced using various methods e.g., functional style, but the most commonly used and efficient way is the use of model or state approach. The steps in writing the Z, involve where an engineer starts by introducing basic sets. These basic sets will not include the details initially but will be defined at a later stage. To make the specification more readable, additional operators are introduced. The next step is to define an abstract state. The abstract state is defined by sets, functions, relations and sequences.

When building the abstract state, an initial state is specified, where the state will change depending on system operation, which will be the before state and the after state. Depending on the system operations the after state can be the same as that before the state. Predicates define what each operation should do, which includes the inputs and the outputs of that operation. Operators may include preconditions, where the responsibility is on the programmer to make sure that those pass before the operation can be executed.

The last step is to validate and verify the design, which is achieved by doing a state and prove the theorem of the system. The process assists in finding errors in the design before the system is actually implemented. Formal specification work as a reference point for all the aspects of the software process that is for eliciting the requirements, the implementation of those requirements, testing of the system, and

developing instruction manuals of the system (Bowen, 2016). With the aforementioned, the specification must be validated or tested in order to ensure that there are minimal errors, which also affect the other parts of the software process.

The specification can be large, which can be difficult to read, and so to overcome this problem, a specification is decomposed into smaller components by the use of schemas (see above figure). Z can be used with other formal languages such as CSP. Work has been done to combine Z with CSP (Benjamin, 1990). VDM specification language is regarded as the direct competitor of Z (Alagar and Periyasamy, 2011). The two are based on first-order predicate logic.

In industry or the commercial world, companies like IBM are known to have utilised Z specifications. According to Bowen (2016), the IBM Customer Information Control System (CICS) had about 2000 pages of Z specification and the designs, with around 37000 lines of code. That said, more work still needs to be done for Z to be made commercially acceptable.

2.9.1 Some of the tools that are used for Z specification

Tool support assists a great deal when developing a Z specification. What makes Z more advantageous than other languages is that Z has a couple of tools to write the specification (Dongmo and van der Poll, 2010).

- CadiZ created by Toyn and McDermid (1995) for formal reasoning
- The Community Z Tools (CZT) by Malik and Utting (Malik and Utting, 2005) (http://czt.sourceforge.net/)
- Fuzz Mike Spivey's type checker for Z (http://spivey.oriel.ox.ac.uk/corner/Fuzz_typechecker_for_Z)

2.9.2 Established Strategy

There are also well-established strategies for documenting a Z-specification such as ES (Established Strategy). Van der Poll and Kotze (2005) also propose an enhanced established strategy for writing a Z specification. As high-level steps on how to write a Z specification are explained above, I proceed to discuss details of the Established Strategy. Wordsworth (1999) provided a great deal of input when it came to the Established Strategy. ES embodies the following sequence of steps:

1. Identify and define basic types and global constants and also describe them

in a natural language:

An example of basic types for an ERP system purchasing module which is discussed in detail in the following chapter would be:

[STRING, AMOUNT, DATE]

[PRODUCT, ORDER, ITEM, CUSTOMER]

2. Then, present the abstract space, the basic types and constants defined above.

3. Provide the current state of the software and also demonstrate that it exists. An example of a current state of the product schema for the purchasing module is given below.

   Requirements statement: "Specify a system that allows a user to view a product that is already in the system, as well as the quantity and the price of each product."

The below schema represents products that already exist in the system:

Product

$$
\begin{array}{|l}
\hline
products : \mathbb{P}\ PRODUCT \\
prodName : PRODUCT \twoheadrightarrow STRING \\
prodPrice : PRODUCT \rightarrow AMOUNT \\
proQuantity : PRODUCT \rightarrow \mathbb{N} \\
\hline
\mathrm{dom}\ prodName = products \\
\mathrm{dom}\ prodPrice = products \\
\mathrm{dom}\ proQuantity = products \\
\hline
\end{array}
$$

The above schema shows the product that is already created in the system, the schema shows the product name, the price of the product and the number or quantity of the same product in the system.

4. Start with a partial definition of each operation and give a short natural language description of it.

   An example of an operation schema for ERP purchasing model is specified in schema *CreateProduct* below:

CreateProduct

$\Delta Product$ *prd*?:*PRODUCT* *nme*?: *STRING* *pr*?: *AMOUNT* *qnty*?: $\mathbb{N}$
prd? $\notin$ *products* (5) *products '* = *products* $\cup$ {*prd*?} *prodName'* = *prodName* $\cup$ {*prd*? $\mapsto$ *nme*?} (6) *prodPrice'* = *prodPrice* $\cup$ { *prd*? $\mapsto$ *pr*?} (6) *prodQuantity'* = *prodQuantity* $\cup$ { *prd?* $\mapsto$ *qnty*?}

5. Determine/Calculate the precondition of operation on the state.
6. Inputs, outputs and correct operation precondition of all operations are tabled.
7. Specify all the schemas that produce an error condition.
8. Use calculus Z schema to make partial operation totals.
9. Assist the reader of the specification by providing additional information e.g., a summary of an operation.

The steps guide the designer or analyst when documenting a Z specification. This sets a standard as to how to write a Z specification. The limitation of Established Strategy (ES) is that it does not provide any guidance about the schema content and the interaction between various operations making up a specification (Van der Poll and Kotze', 2005) The Established Strategy is also not integrated to other well-known design principles and it doesn't take into account some of the Human-Computer Interaction (HCI) principles. ES breaks the rule of visibility from HCI, which instruct: "make things visible to the client".

2.9.3 Enhanced Established Strategy

Van der Poll and Kotze (2005) propose the following steps in order to enhance the Established Strategy:

1. Describe overall global basic types and constants. Encompass all types of which the output is produced to allow for undefined output. Explain in a form of natural language all the types. The improvement compared to the previous strategy is to add all types in the first step.
2. Show the abstract state space, using the constants and basic types stated

above. This step is the same as the previous strategy.

3. Provide an initial state of the system and demonstrate that such a state can be achieved.
4. Show the environment, again using the above constants and basic types (van der Poll and Kotze added an extra component to the environment).
5. Give a definition of each system operations.
6. Define the precondition of each system operation on the state and prove that precondition is explicit in the operations.
7. Stipulate an undo equivalent for every robust operation that changes the state.
8. Stipulate the control module which shows when each user-level operation (which is also a robust operation) is invoked.
9. Specify a table displaying all the robust operations with their inputs, outputs, preconditions for correct operation and error cases.
10. Give more information that will help the reader of the specification.

The enhanced Established strategy was used as a guideline when writing a formal specification in the next chapter.

It becomes a challenge using Z to integrate parts of a different system for a different organisation. One organisation may have written all their requirements in MS-word processor and the other company using Z tool LaTeX. The backlog arises when you ask another company to learn Z, and how to use the tools which, in turn, increases the costs of the project (Bowen and Hinchey, 2012).

2.10 Formal Methods Myths

It is not guaranteed that formal methods produce error-free systems, yet many studies have revealed that using formal methods in software development using an object-oriented design has many benefits (Iddiqui, Akhter and Ian, 2014). Even when formal methods have proved to be beneficial in complex mission-critical projects, software engineers are still sceptical about the use of formal methods. Most engineers view formal methods as a mechanism that is practically both hard to understand and to utilise (Spichkova, 2012a). In line with this, there are so many myths around formal methods.

Hall (1990) published the seven famous myths of formal methods, the biggest myth being that the use of formal methods can assure that the resultant software is perfect, i.e., that it will be without any errors.

Hall (2007) identified and discussed seven myths of FMs, where Jaspan *et al.* (2009), revisited and discussed these:

1. formal methods give assurance that the system is perfect: using formal methods guarantees that the software is free from defects. Formal methods can only reduce the number of defects, but doesn't guarantee a perfect system;
2. formal methods are about proving that programmes are correct: the verification of software properties is that the final product will work perfectly;
3. only critical systems benefit from the use of FMs: this is because of the difficulty of use of FMs and has led to the belief that they can only be used for mission-critical systems. However, formal methods can be used in the development of any system;
4. they use difficult mathematics: FMs are based on mathematics which is the reason why most engineers view them as difficult. According to studies by Hall (1990) at Praxis, they found that the discrete mathematics of software specifications can easily be mastered and used;
5. they escalate the costs of software development. The cost of using formal methods can be high but it does help reduce the cost that will be spent on post-delivery maintenance (Sommerville, 2016);
6. they are incomprehensible to clients because of the use of mathematical notation on formal methods clients may find it difficult to read the specification. But formal methods are made up of additional components that can easily be read by clients (Sommerville, 2016); and
7. nobody utilises them in real-world projects: It is viewed that formal methods are only used for academic studies, yet IBM's CICS project (Bowen, 2016) shows FMs are utilised in real-world applications.

In reality, formal methods are not perfect and do not guarantee software that is free from defects. The use of mathematical notation does not help either when it comes to people actually using them for commercial software.

2.11 Disadvantages of formal methods

Formal methods use mathematical notation, which is viewed as difficult to learn, and also makes it difficult for a client to read and understand a given specification. The specification themselves can become intricate, and hard to practise. This statement is backed up by Hall (2007), who states that “this is clearly a challenge: current formal notations are notoriously opaque, and formal methods tools are almost all hard to use.” Parnas (2010) added that the models are often more difficult to read and write than to read and write the code itself.

According to Hall (2007), formal methods are only applied in critical parts of the systems and are not applied in fast-moving software, such as websites. In fast-moving software, failures are tolerated and even expected. It also becomes difficult to describe the GUI of the system, as they are focused more on the system operations than on the graphic. Furthermore, formal methods are perceived as causing delays in the development process. Alsmadi (2017) added that the other factor that makes GUI designers avoid using formal methods is that GUI specifications are difficult to formalise or prove. However, in recent years, frameworks have been developed to try and produce GUI formal specifications.

As mentioned throughout this dissertation, the notations are not standardised and the tools to support formal methods are not readily available. Formal methods are a small part of the solution of system development problems, used in order to realise the full value of formal method they needed to be integrated into bigger software process.

Liu *et al.* (1995) mention that there is a big gap between real-world and formalism, that transforming clients requirements from informal requirements to formal requirements requires serious clarification of the problem. There is no accepted principle or guidance of eliciting client requirements, and how to specify them using formal specification language. The specification may be accurate, but not correct according to users’ requirements.

In big systems, the formal specification becomes hard to read, write, and most importantly, to be understood by the developers or engineers. Formal methods also become challenging when integrating with current software development techniques (Gabbar, 2006).

2.12 Slow Adoption of FMs in the Commercial World

Most software development companies do not consider it cost-effective to apply formal methods in their software development processes (Crepaldi, 2005). One of the stumbling blocks in the use of FMs in the commercial world is that of perception that formalisation is difficult, and the creation of formal methods is error-prone and time-consuming (Atlee et al., 2013). Hall (2007) differs with this, asserting that formal methods are based on mathematical notation, which is the reason they are perceived as difficult; however, in reality, the notation can be easily learned and used. Bowen (2016) added that it is easier to learn notation than learning a new programming language.

Another reason for slow adoption is that most engineers' views of formal methods as a mechanism that is practically hard to understand and utilise (Spichkova, 2012a). The commercial world or businesses are of the view that the use of FMs can increase the costs of software development due to the level of training that is needed. Education plays a major role in an individual developing and designing the systems, where in addition, management needs to be educated if they are to successfully apply formal methods within their organisations (Bjørner and Havelund, 2014).

> ***Proposition (Prop) 1:*** *Education plays a major role in formal methods adoption. This includes educating from the high school level to the university level as well as organisational training in the use of formal methods. Such education plays a pivotal role in the adoption framework.*

As more software development processes gain popularity, for example, the Agile methodology, there is the view that formal methods do not support other software development processes. According to Dongmo (2011), formal methods can be beneficial in every step of the software development life cycle, as they help in alleviating incomplete and unrealistic requirements at the beginning of the development process, leading to the production of a high-quality product with fewer defects.

Lack of easy step-by-step guidelines regarding how to use formal methods also contributes to the slow adoption. Many developers view formal methods as limited to academic projects for tertiary education. Bowen and Hinche (1995) felt that standards, tools, and education would "make or break" industrial adoption, while

Glass (1996) saw a chasm between academics who “see formal methods as inevitable”.

Most traditional software development techniques are established, and proper standards have been set. Tools supporting those techniques are widely accepted and used in business. On the other hand, formal methods appear to have inadequate tool support. Certain formal methods tools do not work suitably with the development/programming tools. Formal methods tools are also not seen as being user-friendly.

When compared to traditional techniques, there are many certifications that one can acquire and many institutions offering training around those techniques. According to a study done by Davis *et al.* (2013) slow formal methods adoption may also be attributed to certification authorities not having enough education regarding how to appraise formal methods artefacts, and they are not highly informed of formal methods benefits and underlying techniques.

> ***PROP 1.1*** *In addition to the above proposition, formal certificates and diplomas in formal methods ought to be created and awarded to those who qualify. Certification authorities should be well-informed about the benefits of formal methods.*

Normally, when developing a system for clients, users review and sign off the requirements specification i.e., Business Requirement Specification (BRS) or Functional Requirement Specification (FRS). The review is to make sure that all user requirements are included in the specification. The specification can then be used to bill an external client, where, for an internal client, an agreement could confirm that the stated requirements will be developed (see Figure 2 discussion above). Clients find it difficult to review formal specifications due to the mathematical notations used, this results in project delays.

There is also a psychological and human resource factor with the slow adoption in business. Within the organisation or business, some people just do not like formalisms; the same applies to formal methods as some engineers especially those who are already working in an agile environment, will be more reluctant to use formal methods. In business, the development of some projects are relatively fast, so there is little time to conduct a proper formal analysis. Nowadays, individuals change positions frequently, for example, from a software engineer to a manager, or

changing companies. This results in having to upskill new employee, which is time-consuming.

> ***PROP 2:*** *Buy-in from all the business stakeholders is necessary for FM adoption. Getting Top-level management to agree to and accept the use of formal methods may well result in the whole organisation adopting formal methods.*

There are many misunderstandings with formal methods, leading to slow adoption in business. Businesses view formal methods as a technique that places too much emphasis on the theory, rather than real-world applications of FMs. Another huge misconception is that if an FM is used, then there is no need for testing. This ties in with one of the seven myths, where the use of a formal method does not guarantee that the resulting software or system is perfect.

Sommerville (2005) also indicated four reasons why there is a slow adoption from the commercial world:

1. the utilisation of other system engineering techniques i.e., configuration management and structured techniques has improved software quality;
2. software these days is developed and delivered fast the main focus is time to market than quality, where some customers will accept software with some errors if it can be delivered rapidly. Rapid software delivery does not work well with formal methods;
3. the narrow scope of formal methods does not cater for user interface design and user interaction.
4. Lastly, developing formal specifications for system upgrade becomes a time consuming and costly process in which the commercial world is not willing to entertain.

> ***PROP 3:*** *Widely accepted principles and guidelines on FMs can improve the adoption thereof. Practical, real-world examples of FMs successes and failures must be published in the software engineering and management communities.*

2.13 Challenges with Current Development Processes

There is a continuous development of software engineering techniques, tools and methods, as the problems relating to software development have been around since the start of computer systems (Crepaldi, 2005). We will first give a brief description of various types of software development process. A summary table of each development process, along with its strengths and weaknesses, will be presented. Table 2-2 was adopted from *Schach* (2011) *Object-Oriented and Classical Software Engineering 8th Edition book.*

Waterfall life-cycle model: presented by Royce (1970). This is a document-driven approach, where each phase is completed, and a document needs to be produced. Working software is produced later on in the life cycle. This is the most widely used model. It comprises of the following steps: 1) requirements; 2) analysis; 3) design; 4) implementation; 5) post-delivery maintenance; 6) retirement.

Evolution-tree model: the sequence of the steps that need to be followed or executed when producing or maintaining software. The evolution-tree model can also be considered a simplified version of the waterfall model but is closely related to the iterative-and-incremental model. In this model, engineers view the development of software as a maintenance process, constructed on the tree of decisions. Made at different times within the development process, these decisions are influenced by a change requirements or a change request as they are issued (Tomer and Schach, 2002).

Iterative-and-incremental in the real software development world, the analysis phase is spread though out the life cycle and is not done in a single step. The basic software development is iterative, meaning software gets developed in increments. Iteration and incrementation are used together, and there is no one “requirements phase” or “design phase”, but there are multiple occurrences of each phase.

Rapid-prototyping life cycle: to build a rapid prototype and allow clients to interact with rapid prototypes. Then, a requirement specification document is written once the client is happy with the porotype. This improves confidence that the product will meet client requirements. Also, this model allows the design team to gain an understanding from a rapid prototype.

Open-source life cycle-model: has two informal phases, where firstly one developer will build a first version of the system and makes it accessible via the internet or forums, whereupon volunteers can build onto it. The software can then be moved to the second phase, which is post-delivery maintenance. In an open-source project, there are usually no specifications and no design. Code is made available for anyone.

Synchronise-and-stabilise life-cycle model: developed by Microsoft, where requirements are elicited with a potential customer, after which the specification is written. After the specification, they then divide the project into builds. At the end of each the day, the team synchronise (test and debug), and at the end of the build, they stabilise (freeze the build).

Spiral life-cycle model: if all risks cannot be mitigated, the project is instantly cancelled. Developers must be trained in risk analysis. Based on the distinctive risk patterns of a given project, the spiral model guides a team to adopt components of one or more development process models, such as incremental, waterfall, or rapid prototyping (Boehm and Turne, 2015).

Agile processes: governed by the agile manifesto. The common practice of this model is the daily meetings. The main focus is delivering working software over documentation, fast response to requirements changes, and as well as customer collaboration i.e., business stakeholders and coders collaborate on a daily basis for the entire project. The most efficient and effective method of exchanging information to and within a development team is by face-to-face discussion (Beck et al., 2001). Some Agile models use scrums. Scrum depends on self-organizing, cross-functional team system features are delivered according to sprints.

The following table summarises the strengths and weaknesses of each model.

Life Cycle	Strengths	Weaknesses
Evolution-tree model	• Closely models real-world software production. • Equivalent to the iterative- and-incremental mode.	Proper planning is required.
Iterative-and-incremental life cycle	Closely models real-world software production underlies the unified process.	• It requires decent planning and design • The total costs can be higher than the waterfall model • Needs a clear and comprehensive description of the entire system before it can be broken down and built incrementally
Code-and-fix life-cycle model	Fine for short programmes that require no maintenance	Totally unsatisfactory for nontrivial programmes
Waterfall life-cycle model	Disciplined approach document-driven	• Delivered product may not meet the client's needs • The client is unlikely to understand the technicality of documents • No working software is developed until the late during the development cycle
Rapid-prototyping life cycle	• Ensures that the delivered product meets the client's needs. • Design team gains insight from rapid prototype.	Not yet proven (beyond all doubt).
Open-source life-cycle model	Has worked extremely well in a small number of instances	Limited applicability, usually doesn't work
Synchronize-and-stabilize life- cycle model	• Future users' needs are met. • Ensures that components can be integrated successfully.	Has not been widely used other than at Microsoft.
Spiral life-cycle model	Risk driven	• Can be used for only large-scale, in-house products. • Developers have to be competent in risk analysis and risk resolution.
Agile processes	• Works well when the client's requirements are vague. • Visibility of project details increased. • Increased team productivity. • Ability to adjust to	• Appears to work on only small-scale projects. • No emphasis on solution design and documentation. • The project can easily go off track.

Life Cycle	Strengths	Weaknesses
	changes. • Ability to scale.	

Table 2-2 Strengths and Weakness of Different SDLCs

Source: Schach (2011) *Object-Oriented and Classical Software Engineering*

2.14 How formal Methods can help alleviate some of the current problems

Real-world software development projects do not really follow a step-by-step process i.e., from analysis, to design and implementation. There is always an overlap when it comes to these steps.

It is difficult to get the customer requirements right and to complete at first hand. This impacts negatively on other phases of software development, as their artefacts are based on requirements. For example, the design document will be wrong if the requirements are not captured correctly. A formal specification can overcome this, as it allows the engineers to rigorously analyse the requirements and detail properties about the system. This reduces errors and oversight of the requirements.

With the traditional waterfall, one step needs to be completed before moving to the next. In the process of waiting for one step to be finished, the technology is also changing. By the time the project is finished, the technology is already outdated. A design document can be produced from the formal specification, producing two specs at the same time. Sommerville (2005) proposes that there is the possibility of automating the formal specification, such that the code can be produced from it. By having a formal requirement specification, which can also work as a design document, formal methods can fast track the development of those artefacts.

If there is a change in one stage, for example, requirements document, this can also impact subsequent stages such as the design, leading to project delay and an increase in cost. If formal methods are used, they result in minimal changes in the requirement, due to how much of the work has been done in the specification stage. The formal specification also guides the tester in identifying the correct test cases. Test cases can be written directly for the formal specification. This reduces time and costs. Several techniques for stimulating Z utilises Prolog, with two main methods, viz. programme synthesis, and structure simulation (Dongmo, 2016).

The Standish report in Hastie and Wojewoda (2015) indicated that only 29% of the projects in a traditional software process are delivered successfully. The other 52% of the projects are either delivered late, or they do not meet customer requirements. Lastly, 19% of the projects are projects that have either failed or discontinued. By critically analysing the requirements with formal methods and reducing requirements ambiguity, this has the chance of increasing the percentage of projects delivered that meets the client's requirements.

2.15 Formal Methods in Practice

Since the development of formal methods in the 1980s, their adoption or use within the business arena is slow (Davis et al., 2013). However, the following software and hardware giants are known to be using formal methods:

- Amazon;
- Intel;
- NATS;
- Xilinx; and
- NASA.

Other companies known to also use FMs are: Qualcomm, Nvidia, Cisco, Broadcom, Samsung, Mediatek, AMD, and Huawei. Google and Microsoft's main focus was software, but they are starting to develop their own hardware, and they are also adopting formal methods (Cousineau et al., 2012). Start-ups are slowly picking up formal methods as this provide a good return on investment (ROI) with clean code, meaning that less money is spent on rectifying defects.

Next, we elaborate on the successful use of FMs by the mentioned companies.

INTEL

Intel's core business is hardware, where for hardware to work, the following needs to be developed: Microcode, Firmware, Protocols, and Software. In almost all the products, Intel experience problems with the diversity of verification (Fix, 2008). According to Harrison (2010), Intel developed various solutions trying to solve verification problems. Their solutions include propositional tautology/equivalence checking (FEV), symbolic simulation, symbolic trajectory evaluation (STE), and temporal logic model checking.

Intel experienced numerous problems with their products, the most challenging was a physical problem with the overheating of their Chips, and the FDIV bug, which could be solved through the use of FMs. Intel invested over $147 million to cover the cost incurred from chip overheating and the verification problems that led to the improvements of FMs within Intel. Intel has realised numerous benefits with using formal methods, and they continue to use them on many projects (Harrison, 2003, 2010).

AMAZON

Amazon is an online shopping giant that utilises formal methods. Amazon is the largest internet-based retail business in the world by sales and market capitalisation. According to Newcombe (2013), Amazon's software engineers started using formal methods, mainly for formal specification and model checking in 2011. Their main aim was to solve design problems in their critical systems. Amazon tried to use different techniques in order to minimise defects in their system but still discovered many defects hiding in their critical systems. Some of the techniques tried were code reviews, static code analysis, and traditional testing, e.g. stress testing. The main reason for failure in these techniques was human error.

To solve the above challenges, Amazon embarked on the use of FMs. They did not develop their own FM software but looked for an off-the-shelf Method, which would yield high returns on investment. They started using a formal methods specification language called TLA+ created on predicates and basic set-theory. TLA+ falls under the Axiomatic type of formal methods (Cousineau et al., 2012). Most engineers within Amazon were familiar with TLA+, which was a major advantage, as they did not have to spend money and time training their staff. The main benefit of TLA+ is that it describes the preferred correctness (the what, business/user requirements, etc.), of the system, and the design of the system (how, functionality) (Newcombe et al., 2015).

Amazon adopted the use of TLA+ on 10 large complex systems, and in every system, they have realised many benefits. Amazon was able to discover defects that they were unable to find beforehand, as well as gain a thorough understanding of the system that enabled them to make huge performance optimisations, without sacrificing correctness. The buy-in from senior management and the technical team leaders helped to speed up the adoption of formal methods within Amazon, in which some team members taking up to 3 weeks to learn TLA+ from scratch (Newcombe

et al., 2015).

Formal methods have been a big success at Amazon. They have assisted in preventing serious bugs before the system goes to production, and they have helped to increase productivity and innovation.

XILINX

Xilinx has also adopted FMs to improve the communication between its software and hardware. Xilinx is an American company that develops, designs, and sells programmable logic products. These include software design tools, integrated circuits, design services etc.

Xilinx, together with the University of Kaiserslautern and One-pin Solution, partnered on a project to investigate how to apply formal techniques to the verification of a Xilinx soft IP core product that is comprised of firmware and hardware components (Xilinx, 2012). They found out that it was possible to capture the interaction of firmware and hardware in a scalable formal-verification environment. This joint venture between business and academia was based on a type of formal method called interval property checking.

IPC falls under bounded model checking, which is a Model-based category of formal methods, limiting the scope of properties to a number of clock cycles, using Bo Boolean satisfiability (SAT) solvers to perform the actual model checking. IPC differs from other models by allowing the window of clock cycles over which a property may be asserted to start at a random point in time. The use of formal methods brought numerous benefits within Xilinx, and has also increased confidence in the functional correctness of their SEM core, and has Xilinx's continued commitment to quality IP distribution (Xilinx, 2012).

NATS

NATS is a UK-based company, which specialises in air navigation software. According to Carlier, Dubois, and Gotlieb (2012), NATS handled 2.2 million flights in 2009, covering the UK and eastern North Atlantic. NATS has developed a tool called iFacts (interim Future Area Control Tools Support), which provides controllers with a set of tools that enables them to increase the amount of air traffic they can handle. iFacts also has the following capabilities: prediction, deviation alerts, and conflict detection.

When developing iFacts, NATS adopted the use of FMs. The system was developed using Z for functional specification, Maths for algorithm specification, State tables for HMI specification, and the rest was natural language, which is an informal technique. As this system was deemed critical, and people's lives would depend on in it, the system had to be set up in such a way that it works correctly, and without any uncertainty.

To successfully implement this system, NATS had to send its engineers to a three-day course for Z notation reader training, where they trained about 75 specialists on how to read Z. They then also enrolled some engineers on another 3-day course on how to write Z, in total about 11 engineers. It took about three months for the engineers to be fluent in Z while on the job, and about one week for readers whilst on the job.

NATS managed to deliver the iFacts system on time, with minimal defects. The use of formal methods increased productivity within NATS. The investment in training assisted training the staff leading to the success of the project.

NASA

NASA is also a major advocate of FMs. NASA has written guides and standards for system development. They recommend the use of formal methods during all stages of the SDLC, but mostly on the formal specification for requirements (Zhang, 2009). In this regard, one should note the 5th commandment of FMs, namely, "thou shalt not abandon thy traditional development methods" (Bowen and Hinchey, 2012).

Other Earlier Successful use of Formal Methods

The companies discussed above manage to successfully use formal methods in their software development. Subsequently, they realised good return on investment, where the number of defects has been reduced significantly, and the systems or products work with minimal ambiguity. Pressman (2009) states that the sooner a defect is found and corrected during development, the cheaper it is to resolve. See Figure **2-3** above.

Other places where formal methods have been implemented successfully includes railway signalling systems (Dehbonei and Mejia, 2012), spacecraft systems (Easterbrook, Lutz and Covington, 1998), and medical control systems (Jacky, 2004), They have also been used for software tool specification (Fenton and Neil,

2000), the specification of part of IBM's CICS system (Wordsworth, 1999).

CICS (Customer Information Control System) was developed using Z, and it was reported that there was about 40% drop in estimated faults, where likewise, the cost of the project has been reduced significantly (Fisher, 1990). Z was also used in specifying the Inmos T800 Floating Point Transputer system, which reduced project cost, as well as the delivery of good quality software (Bowen, 1996). In France, the B-method was used to develop a Paris Metro System (Lamsweerde, 2000).

From the aforementioned, we can see that formal methods show success in the past, where it has also shown success in the present day with companies like Amazon and Intel. FMs remains a viable method for correct software development.

The above discussions lead to the following proposition:

> ***PROP 4:*** *Tools that are readily available and up to date with the latest technology should facilitate the adoption of FMs. Such tools ought to be integrated with the requirements management software and standard software programming tools e.g., MS Visual Studio.*

And a refinement of ***Prop 3*** above:

- ***PROP 3.1:*** *Widely accepted principles and guidelines on FMs can improve the adoption thereof. Practical, real-world examples of FMs successes and failures must be published in the software engineering and management communities. Publications of formal methods successes in terms of cost savings in projects, clear specifications produced, and the overall final product delivered with fewer defects will raise much interest needed for the adoption of FMs. Practical, real-world examples of FMs successes and failures must be published in the software engineering and management communities.*

2.16 Summary

Chapter 2 discussed the ERP system in terms of what it is, and what modules ERP comprises. In addition, the challenges that arise when implementing an ERP system within the organisation were discussed. Chapter 2 then touched on formal methods, placing more emphasis on the Z specification language. Furthermore, the chapter explained the differences between informal and formal methods in a tabular form. Types of formal languages were identified, and the minimal description of Z notion discussed. Myths around formal methods were identified. Lastly, the chapter focused on the reasons why there is slow adoption and suggested ways to fasten the adoption of formal methods in the commercial world. To conclude the chapter, practical examples where formal methods were used in the commercial world received discussion.

From the chapter discussion, we can conclude that formal methods remain a viable software development method to deliver software with fewer errors. They have been successful in the past and remain successful in the present day.

The next chapter focuses on the formal specification of an ERP system, using Z. Case studies will be given and then the specification is written informally using natural language and formally using Z. For Z, each schema is given accompanied by a discussion of what it means. At the end of the chapter, a preliminary framework is presented and explained.

Chapter 3 Informal, Semi-Formal and Formal Specification

3.1 Chapter Layout

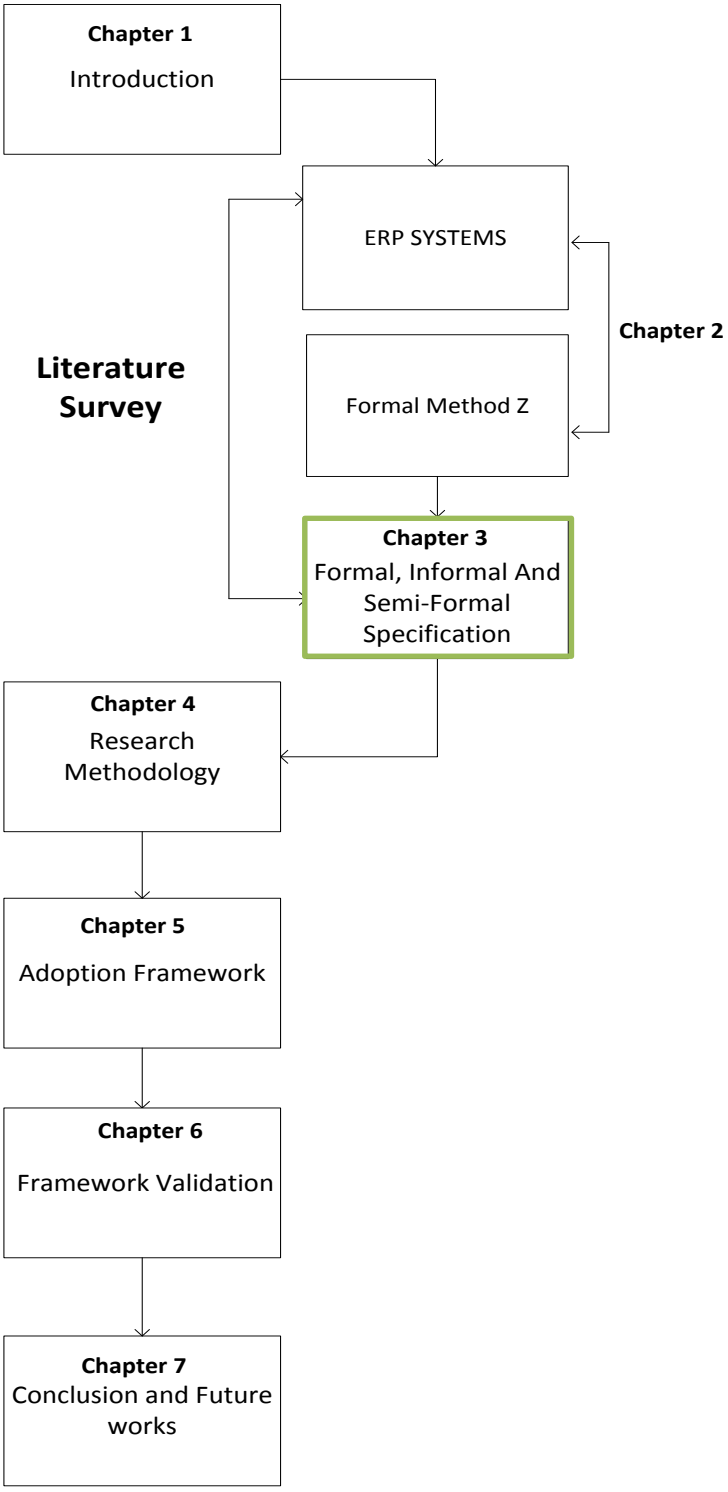


Figure 3-1 Chapter Layout

3.2 Introduction

The previous chapter introduced ERP systems, as well as formal methods. Various modules and the challenges of implementing ERP systems within the organisation were discussed. The chapter then introduces formal methods by placing more emphasis on the Z-specification language. Other types of formal language were identified, and the minimal description of Z notion was discussed. The reasons why there is a slow adoption in business are discussed, where some of the reasons are the expense, as well as being viewed as difficult to understand, due to mathematical notation. Furthermore, it analysed the problem with the current software development life cycle, and how formal methods can resolve those problems. The last two sections of Chapter 2 details preliminary suggestions that were made on the ways to increase the use of formal methods in business, and lastly, the cases where formal methods have been applied successfully in business, in such cases as Amazon, Intel, and NASA.

Chapter 3 will document ERP specification in an informal way and a formal way. This will be structured by providing a case study first, followed by the specification. Parts of the informal specification will be discussed i.e., a UML process diagram, use case diagram, and the details of the process and use case diagram in a tabular format. Before writing a formal specification, a brief introduction to mathematical set theory is presented. The last section of this chapter presents a formal specification in Z for the purchasing module of the ERP. To close off the chapter, a preliminary framework is presented.

3.3 Requirements Specifications

Formal methods specification is linked to design in many ways. From the specification itself, a design can be derived. The development of the specification is an incremental process, this requires the engineer or the writer of the specification to make detail system analysis, that in most cases will uncover errors and discrepancies in the informal requirements specification. Using FMs allows a software engineer to ask questions that may be postponed until the implementation phase (Krause et al., 2012; Wing, 1990). The below diagram shows the relationship between formal specification and other artefacts of the development process:

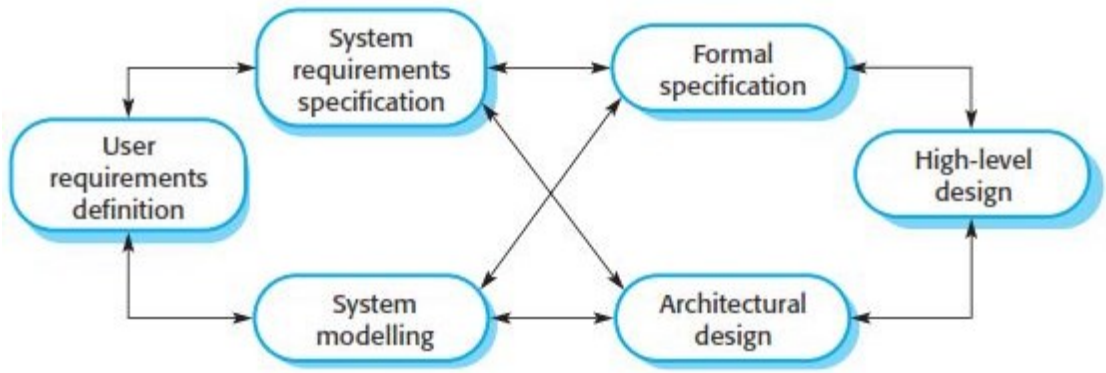


Figure 3-2 Formal specification in the software process (Sommerville, 2016)

From the diagram, we observe that the specification and the design can be carried out in parallel. The user requirement definition is written in natural language, which also feeds into other artefacts of the formal methods development process. System requirement specification can then be developed along with system modelling. The formal specification feeds into the system modelling and the high-level design.

According to Hall (2007), the prevalent argument of using formal methods is error findings at an early stage of software development. The major cost of developing formal specification is the time required for engineers to understand system requirements, decide on the appropriate method to specification and developing a formal model of the system (Crepaldi, 2005). The reduction of the costs happens in the later stages of system development. This results in less work in correcting requirements, and less error correction when it comes to system testing. The following graphs represent the cost of software when informal methods are used, and the costs when formal methods are used.

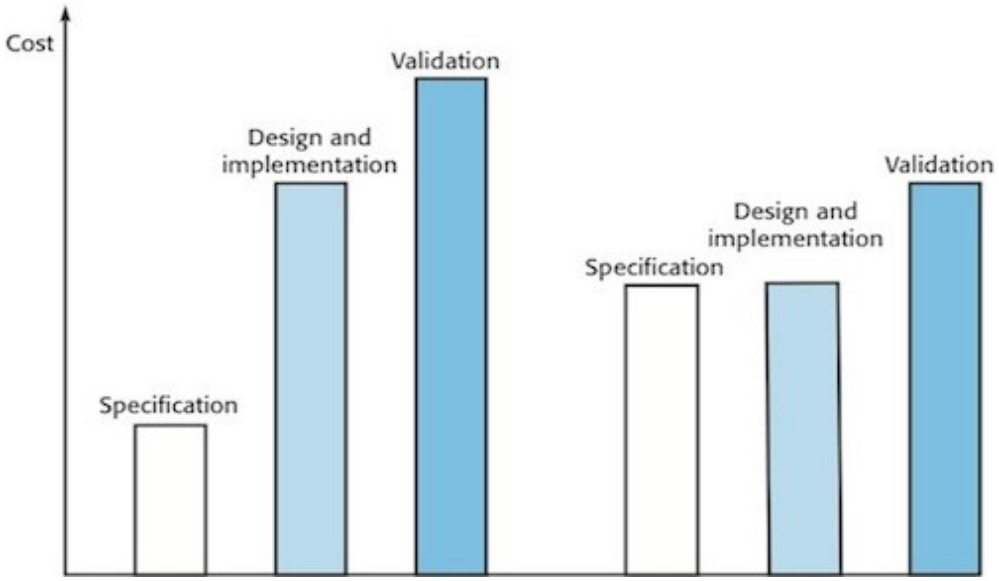


Figure 3-3 Software development costs with formal specification (Crepaldi, 2005)

Figure 3-3 shows the costs of the system development process. The first three bars of the graph show traditional development process costs and the last three bars show when FMs are used. When using traditional methods, about 50% of the costs are attributed to the validation of the development cost. Furthermore, the design and implementation costs are double the cost of the specification itself. When FMs are incorporated into the software process, the formal specification costs and the implementation and design costs are almost similar, while the validation costs have been reduced significantly. The graph shows the total costs of the software development process while using formal methods is less when compared to traditional methods.

Davis (2013), mentions the essential properties of specification document, which are: correctness, completeness; unambiguous meaning (there must be no interpretation); precision i.e., it should have only the necessary information; verifiable and traceable in a way that it should be tested and all the requirements must be linked to other components of the system; the specification document should be independent of design; consistency should exist, where there ought not to be conflicting features, and it ought not be comprised of irrelevant features; where the last property is annotated (this applies mostly when using the Z specification).

Next, I introduce aspects around ERPs through a case study.

3.4 Case study

The following case study is written based on the experience of the researcher and the company names are made up for illustration purposes.

A company (#Parts) in Johannesburg South Africa wants to implement an ERP system. The company is a manufacturing factory which produces car parts for sale. These parts are then sold to different car dealer and service stations around the country. #Parts has around 50 employees. Currently, it is a manual and cumbersome process when employees apply for leave. Files and paper trays get lost within the company, it is also difficult to track which employee applied for leave, and which leave was approved. Reporting becomes tedious, and leave balance is not tracked properly.

The management board resolved to implement a mini-ERP system, mainly on the HR module. They have decided to start with the leave application feature. A member of management suggested that they use formal methods for this project, in order to minimise system errors and project delivery timelines.

3.4.1 General ERP informal requirements

In this section, requirements will be presented in an informal way, viz. natural language. The requirements relate to the company above in the case study (above requirements definition). The focus will be on the HR module of the ERP system, specifically on the employee leave functionality. The informal requirements will be tabulated.

3.4.2 HR Module Requirements

The below table list the informal requirements of the ERP system. These requirements are for the HR module focusing on employee leave.

Requirements No	Description
1	Users must be able to capture employee information. The following information must be captured: • Name and Surname • Designation • Contact details

Requirements No	Description
	• Date of birth
2	Users must be able to apply for leave. The following details the type of leave a user can apply for: • Annual leave • Sick leave • Family responsibility leave • Study leave • Maternity leave
3	Leave application must be approved by a manager.
4	User must be able to view leave balances.
5	User must be able to view and download a payslip.
6	User must have the ability to capture performance reviews.
7	The system must keep track of time and attendance of an employee.
8	All training planned and attended by an employee must be recorded on the system.

Table 3-1 Leave Application Requirements (synthesised by the researcher)

Above are the HR requirements in natural language, which is English. The listed requirements can be interpreted in different ways and can cause a great deal of confusion and ambiguity. IIBA (2012) states that requirements must be SMART, meaning that a requirement must be Specific, Measurable, Attainable (which is it should be achievable and actionable), a requirement must be Realistic, and lastly, a requirement must be time-bound (which is Traceable and Timely).

To clarify requirements in a traditional software development model, business processes are developed or mapped. For this dissertation, I use an example of a leave application business process. This a UML diagram mapped using BPMN (Business Process Mapping Notation), defined as follows: “Unified modelling language (UML) is a graphical language used to stipulate, virtualise and document the properties of software” (Coates, 2012).

3.4.3 Unified Modelling Language

UML stands for unified modelling language, which can be described as an object modelling language that unitises several diagrams to model a system. These illustrations can be used at different stages or sections of the specification to present the system components (Ma, 2008). UML also has a formal component, called Object Constraint Language, which defines the rules that ought to apply to UML. UML is considered easy to use and supports numerous development methods (Sengupta and Bhattacharya, 2006). UML can define the following type of (Scott, 2000).

- Use case diagrams (see Figure 3-7 USE Case Diagram Inventory System)
- Process diagrams (see
- Figure 3-4 Leave Application Process Diagram (synthesised by the researcher))
- Class diagrams (see Figure 3-8 Class Diagram Inventory System)
- Sequence diagrams
- Deployment diagrams
- Statechart diagrams
- Collaboration diagrams

A leave application process is depicted below.

3.4.4 Leave Application Process

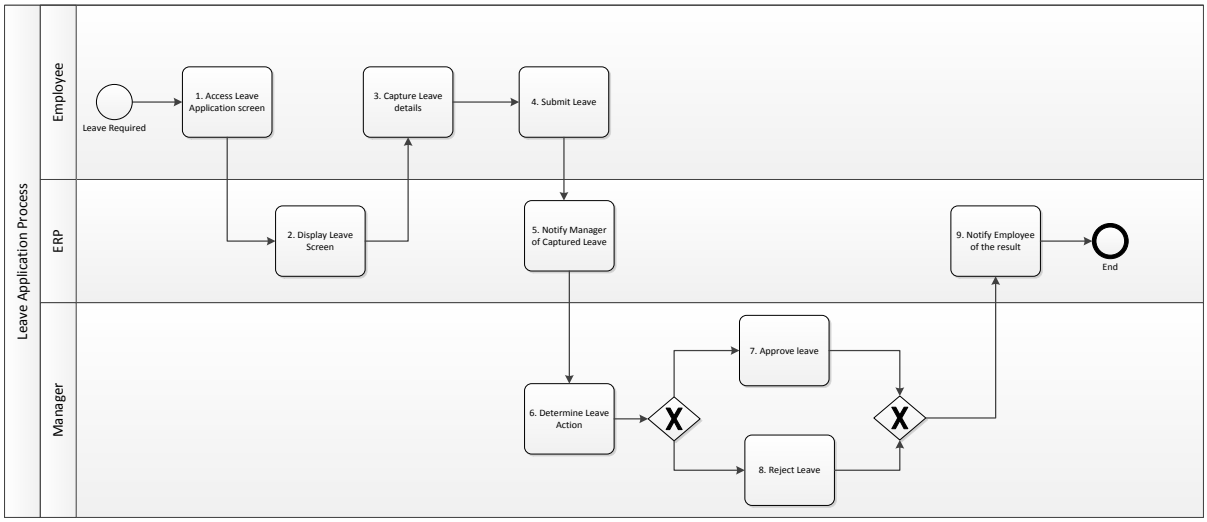


Figure 3-4 Leave Application Process Diagram (synthesised by the researcher)

The above process diagram shows the steps that need to be followed when an

employee is applying for leave. The diagram shows all the actors that are impacted by this process i.e., employee, the ERP system, and the manager who does the application rejection and approval. The next table explains each process step in detail.

The above process diagram is composed of the following notations.

Swim lane

A Swim lane displays a role that is responsible for performing a specific task

Start

The start represents the trigger to the process.

Task

The Task represents the actions that need to be taken by a specific role.

OR

OR/ executive symbol represents what tasks must be performed after a decision has been made.

End

End symbol represents the final step of the process or the end of the process

One mistake that is linked with the business processes is that business tends to

automate old, mostly ineffective processes, which as a result do not see any improvement. Hammer (2003) encourages that in order to archive real organisational improvement, technology must be used to redesign business processes.

3.4.5 High-level process description

Process Description	
Actors	The actors of this process • Employee • Manager • ERP system
Business Rules	• *Sick leave can be backdated* • *Leave must be approved within 3 days of application* • *Application is allowed to go to negative days of up to 3 days.*
Step	**Description**
1.	Access Leave Application Screen – the employee navigates to the leave application screen on the ERP system.
2.	Display Leave Screen - the system displays the leave application screen.
3.	Capture Leave Details - employee selects the leave type they wish to apply for, also the “start date” and “end date”.
4.	Submits Leave – employee submits the leave to the manager for the manager’s approval.
5.	Notify Manager of Captured Leave - the ERP system notifies the manager of employee’s leave, this can be in the form of an email.
6.	Determine Leave Action – the manager can determine whether to accept or reject leave.
7.	Approve Leave – Manager approves employee’s leave.
8.	Reject Leave – Manager rejects employee’s leave.
9.	Notify Employee of the Result – the system notifies employee on the status of the leave. **If** approved, an employee will receive an email informing

	him/her of approval, **Else if** Rejected, the employee also receives a notification.

Table 3-2 Process Description (synthesised by the researcher)

The above table explains the steps of the business process as mapped. The table also includes the business rules applicable to applying for leave on an HR system. IIBA (2015, p.33), defines “business process an activity or set of activities that will accomplish a specific organizational [sic] goal”. It’s also a simplified view of the organisation.

3.4.6 Use Case Diagram

A **Use case diagram** can be defined as a graphical presentation of how the system operates, as well as the actors who interact with the system. Use cases are part of Functional Requirement Specification (FRS), which describe what the system should do. Use cases are limited to functionality that is externally visible to the user of the system (Kotonya and Sommerville, 1998). Use cases are also inadequate when describing non-functional requirements (Sengupta and Bhattacharya, 2006). Non-functional requirements are qualities that are important to the system, not the behaviour of the system, and these include usability, reliability, scalability etc. (Nathan and Scobell, 2012).

When modelling a use case, the following activities are recommended. Firstly, it is necessary to identify the actors who are going to interact with the system, after which, you are required to identify individual use cases. Lastly, the relationship between the actor and the use case is indicated. A use case diagram and the above business process are mapped using Unified Modelling Language (UML). Use cases are written in natural language, which makes them easy to understand, and acceptable to a customer, as opposed to formal methods. However, because they are written in natural language, they are open to interpretation and misunderstanding. Use cases can also be incorporated into other aspects of software developed, such as costs estimating, project planning, and user manuals.

The below diagram is the Use Case Diagram of the HR Module within the ERP system, the component presented is of leave application.

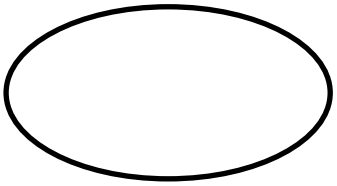
A use case diagram has the following components (Moremedi and van der Poll,

2013, 2019):

- **Actors (stick man)** – represent a role that interacts with the software. This can be other software or an actual person.

Observation: The researcher recommends that "stick man" could in future versions of UML be replaced by a gender-neutral figure.

- **Use case** – is an oval shape that represents functions of the software.



- **Lines** – indicates the relationship between actors and use cases.

The following diagram presents a use case diagram. It indicates the employee will interact with the system. The Use Case diagram shows that the employee can log-in to the system, apply for leave and the manager can approve and reject the leave.

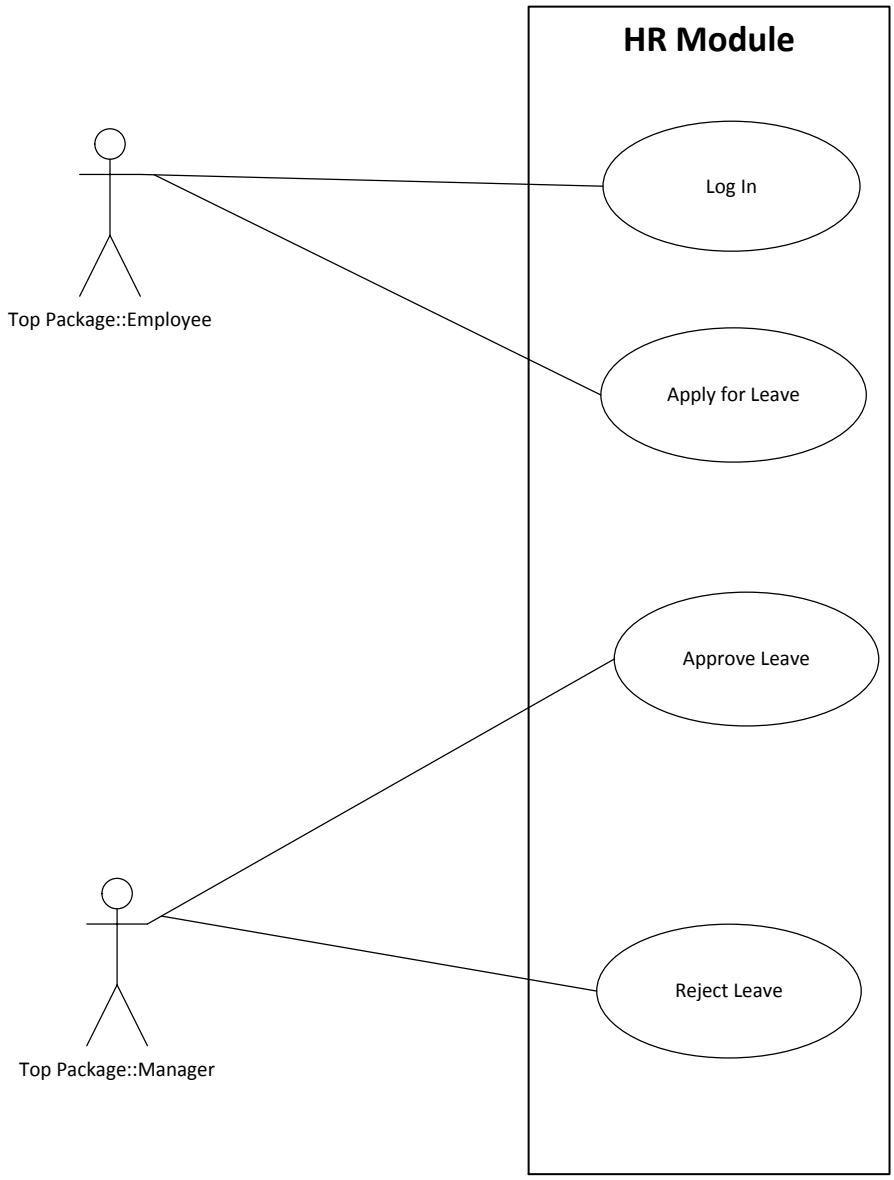


Figure 3-5 Use Case Diagram

The above use case diagram shows all the functionality that the employee and the manager can perform on the system. The use case is further described in a table format below.

3.4.7 Use Case Model Description

Table 3-3 gives a description as to how the user and system interact when applying for leave. The table first defines the pre-condition and the post-condition of the use case. Business rules are also documented. The table expands more on the process diagram, and the requirements table above by showing how the requirement will be fulfilled.

UC - 01	**Apply for leave**
Brief Description	**Begins once** a user wishes to apply for leave **Involves** the user capturing and submitting leave application **Concludes once** When an application notification has been sent to the manager
Preconditions	User is authorised to apply for leave online
Post-conditions	• Application successful or unsuccessful • Application notification sent to manager
Actors	Primary • Employee
Business Rules	• User must be able to check leave balance • User must be allowed to go negative three days • User must be able to perform a backdated leave application.
Triggers	• The employee wishes to apply for leave
Flow of Events	
Basic Flow	

User Action		**HR System Response**
1.	Access the leave application screen.	Displays the leave application screen.
2.	Select the leave application dropdown list.	▪ Displays the types of leave o Sick o Annual o Family responsibility o Maternity leave
3.	Choose the type of leave	None
4.	Enter “Start Date”	Calculate the number of

	and “End Date”	days
5.	Submit Leave	Validate the type of leave and the number of days entered. *If leave balance is negative 5 follow Step 7* *If leave balance is correct follow Step 6*
6.	None	Display leave application sent to manager message for approval
Alternate Flows		
Application Unsuccessful		
User Action		**PRMS Response**
7.	None	Display leave application unsuccessful message
8.	None	Enable the *new* button in the edit panel

Table 3-3 Use Case Description (synthesised by the researcher)

Figure 3-5 Use Case Diagram shows the interaction between the employee and the system when applying for leave. Table 3-2 then gave a detail description of the diagram. For this dissertation only, leave application will be shown as a use case. Next, the formal specification will be presented for the purchasing model of the ERP system.

3.5 Formal Specification in Z

The Z specification will be guided by the Enhanced Established Strategy, as outlined in the previews chapter. The principles suggested by Kotze and Van Der Poll (2005) will also be incorporated where possible in the construction of the specification document. This specification will describe *what* the system must do, not *how* it is going to do it. The specification will work as a single point of reference for the requirements analyst, programmer/developer, the tester and trainer or a person who will write the system manual (Spivey, 2010). It should also be noted that Z is not suitable when specifying synchronized operations; Z is most suited for sequential operations (Boca, Siddiqi and Bowen, 2010).

For one to be able to write a Z specification, they need to have knowledge about set theory. The section below will give a high-level explanation of the set theory.

3.5.1 Set Theory

Mathematical set theory notion has existed for a very long time. Enderton (1977) regards George Cantor as the father of set theory. Gottlob Frege further published a book around 1893 and 1903 demonstrating how maths can be created from the values of the set theory. Then, Russell's paradox was created from Gottlob Frege's set theory i.e.,

$$A = \{x \mid x \notin x\}$$

This reads as follows: Set A is defined as the set of all elements x, such that x is not an element of itself. There is an inherent contradiction in that A contains itself, where, if this is true, then by the description it is not a member of A. Conversely, if A does not contain itself, then by description, it is a member of A.

In 1908, Zermelo Ernst suggested the structure of axioms for set theory. This gained many critics in the mathematics world. Abraham Fraenkel added to this work by introducing the replacement axioms. A total of 10 set theory axioms where developed, and became known as the Zermelo-Fraenkel axioms (Enderton, 1977). Below, we briefly present some introductory set-theoretic ideas.

Sets

A set can be described as a container, where the items inside the container are called elements. Furthermore, a set X is a (finite or infinite) unordered assembly of mathematical objects called elements of the set.

Example

Consider the set S = {1, 2, 3, 4, 5}, a subset of integers which is the default set for our discussion that follows.

We can say 1 is an element of A, i.e.:

$$1 \in A$$

From the definition of S, we can also say that $5 \in S$, which reads: 5 is an element of S. From the definition of S, we also conclude that any number that is not in the set (container) is not an element of S, e.g. 6 is not an element of S, i.e. $6 \notin S$.

Infinite set

To show infinite set in set-theoretic list notation, one uses 3 dots:

$$S1 = \{1, 2, 3, 4, 5, ...\}$$

We can use a variable to definite the scope of a set. For example, for the finite subset S = {1, 2, 3, 4, 5} of S1 above we could write S1 in set-builder notation as (Enderton, 1977):

$$S = \{x \mid x > 0 \text{ and } x < 6\}$$

The above reads as: “S is the set of all ‘x’ such that x is greater than 0 and less than 6”. Naturally, the advantage is that this allows you to scale with relative ease.

Empty Set

The empty set is a set that has no elements in it, also known as the “Null Set”. An empty set is traditionally represented by the symbol Ø or simply {}.

$$(\exists x)\ (\forall y)\ (y \notin x) \text{ – ZF Empty set axiom}$$

$$\text{Also, Empty Set} = \{\}.$$

Universal Set

A universal set is a set that contains all possible elements from a designated domain. Traditionally it is symbolised as U. It is easily understood or explained using the Venn-diagram notation. John Venn developed Venn diagrams in 1880 to show logical statements, and the relationships between sets (Bottoni and Fish, 2011). Below is an example of a Venn diagram.

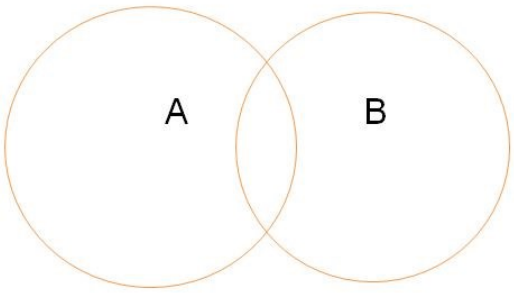


Figure 3-6 Venn diagram

Source: *Drawing area-proportional Venn and Euler Diagrams* (Chow and Ruskey, 2004).

Natural numbers (non-negative integers): These are the everyday numbers we

use to count, and they are represented as:

$$\mathbb{N} = \{0, 1, 2, 3 ,4, 5, \ldots\}$$

Integers: These include the natural numbers as well as negative numbers; in list notation, we could write the set of integers as:

$$\mathbb{Z} = \{\ldots ,-3, -2, -1, 0, 1, 2, 3, \ldots\}$$

Fractions: These are part of the set of rational numbers and they are represented by $\mathbb{Q}$. Real Numbers indicated by $\mathbb{R}$ include the rational numbers and the set of Irrational numbers indicated by $\mathbb{I}$.

Binary Union is denoted by $\cup$. Generally, set-theoretic union is first specified in simple terms for 2 sets (Enderton, 1977), and then it is specified for the distributed case. From the above Venn diagram, we can define set C to be a union of A and b, i.e. $C = A \cup B$. In natural language the set C may also be described as "A or B".

Binary Intersection of A and B includes any values that are contained in both sets and is presented as follows:

$$A \cap B$$

In natural language, binary intersection can be described as "A and B".

The difference between A and B can be described as the values that are in set A but not in set B. The difference is presented as:

$$A - B$$

SUMMARY EXAMPLES

Suppose A = {1, 3, 5, 7, 9, 11} and B = {3, 4, 5, 6, 7}

Then

- The union of A and B: $A \cup B = \{1, 3, 4, 5, 6, 7, 9, 11\}$
- The intersection of A and B: $A \cap B = \{3, 5, 7\}$
- The difference $A - B = \{1, 9, 11\}$

Subsets

Suppose A = {a, b, c, d, e} and B = {a, b, c}

We denote a subset as $B \subseteq A$ reading as B is a subset of A. We can also say B is a proper subset of A, denoted by $B \subset A$ – meaning every element of B is also an element of A, but A contains more elements than B. In a non-typed set theory, a set

can also be an element of another set, mixed with other elements that are (traditionally) not viewed as sets, for example:

Suppose A = {1, 2, 3, {3}, 4, 5}

Then we have:

- {3} ∈ A,
- {3} ⊆ A, and
- {3} ⊂ A.

Infinity: In 1923, Von Neumann proposed that an infinite set contains a mapping from the set of natural numbers to the elements of the set, i.e. the set contains an infinite number of elements (Nerode and Shore, 1997).

Further discussion of infinity is beyond the scope of this dissertation.

Power Set: A power set of a set A, denoted by $\mathbb{P}(A)$ is defined as the set of all the subsets of the given set, e.g.:

If A = **{a, b, c}**, then $\mathbb{P}$ (A) = {Ø, {a}, {b}, {c}, {a, b}, {a, c}, {b, c}, **{a, b, c}**}.

The **cardinality** (indicated by ||) of a set is the number of distinct elements in the set (elements in a set are not duplicated).

Examples

If A = {a, b, c, d, e}, then
Cardinality of A is 5: |A| = 5
If B = {a, a, f, f, f}
No duplicates the cardinality of B is 2: |B| = 2
If D = {∅, {∅}}
|D| = 2
F = {n | n ∈ $\mathbb{Z}$}
|F| is infinite

Regularity: Every non-empty set A has at least one element disjoint from A (Enderton, 1977). (∀A) (A ≠ Ø → (∃x) (x ∈ A ∧ x n A = Ø)) The axiom limits set theory to sets in which the elements of a set must be identified. Some of the consequences of this axiom are (Enderton, 1977; Nerode and Shore, 1997): “No set can be a member of itself, there exist no sets x and y such that x ∈ y and y ∈ x,

There exists no infinite descending sequence of sets e.g. … ∈ f(2) ∈ f(1) ∈ f(0), where f is a function with the domain of the natural numbers."

Proof of the above three properties lies outside the scope of this research. Further details are available in Enderton (1977).

A choice function is a function f, distinct on the assembly X of non-empty sets, in a way that for each set A in X, f(A) is an element of A. With this concept, the axiom can be stated:

Axiom of Choice — For each set X of nonempty sets, there exists a choice function *f* demarcated on *X*.

Formally, it may be expressed as follows:

$$(\forall A)\,(\forall x)\,(x \in A \rightarrow x \neq \varnothing) \rightarrow (\exists f)\,(\text{func}(f) \wedge \text{dom}\,(f) = A \wedge (\forall x)(x \in A \rightarrow f(x) \in x)))$$

Limitations

It becomes challenging to create an automated theorem prover for ZF (Zermelo-Fraenkel) set theory (Steyn and Van der Poll, 2007). However, two prominent software specification languages, B and Z are based on ZF, regardless of ZF having an infinite axiomatisation. In the software industry, the limitations of (automated) proving theorem become challenging when working with set-theoretic proofs arising from a formal specification.

Next, the Z specification language is presented on the strength of a case study.

3.6 Purchasing module formal requirements specification

For the Z specification, this dissertation will focus on the purchasing model discussed next as adopted from Steyn and van der Poll's (2007) work – *Validating Reasoning Heuristics Using Next-Generation Theorem-Provers*.

The procurement model, also known as the purchasing model, enables capturing of orders, and processing of orders. The purchasing model can also be expanded to other functionalities of order fulfilment, such as stock, financial, customer information, and reporting. The scope of this will be limited to order placing and processing. Rules will be provided in tabular format.

No	Description
1	User must log-into the system first before they can perform any tasks
2	The system must be able to keep track of stock for several products
3	The product should have a name, price, and quantity of available stock recorded on the system.
4	Each product must have a unique name.
5	User should be able to update products name, price and quantity of stock on hand.
6	User should also be able to delete products.
7	The system should have the ability to produce a list of all products that are below the threshold.
8	The system should allow for the capturing of orders.
9	Once a new order for a specific product is captured, it will stay on the “pending” status.
10	All orders on the pending status can be deleted, once deleted the status should change to “Cancelled”.
11	The quantity of an order should always be more than one.
12	The record of the quantity, price, and product name order must be kept.
13	All orders with the pending status should be processed if there is enough stock to hand.
14	Once the order is processed, the status should change to “processed”, and the quantity should decrease with the same number of products are ordered.
15	Customer information needs to be stored and linked to the order. Information includes the name, address and phone number must be stored.
16	One customer can have multiple orders.

Table 3-4 Procurement Module Requirements (Steyn and Van der Poll, 2007)

Next, a UML use-case diagram of the process is presented.

Use Case Diagram

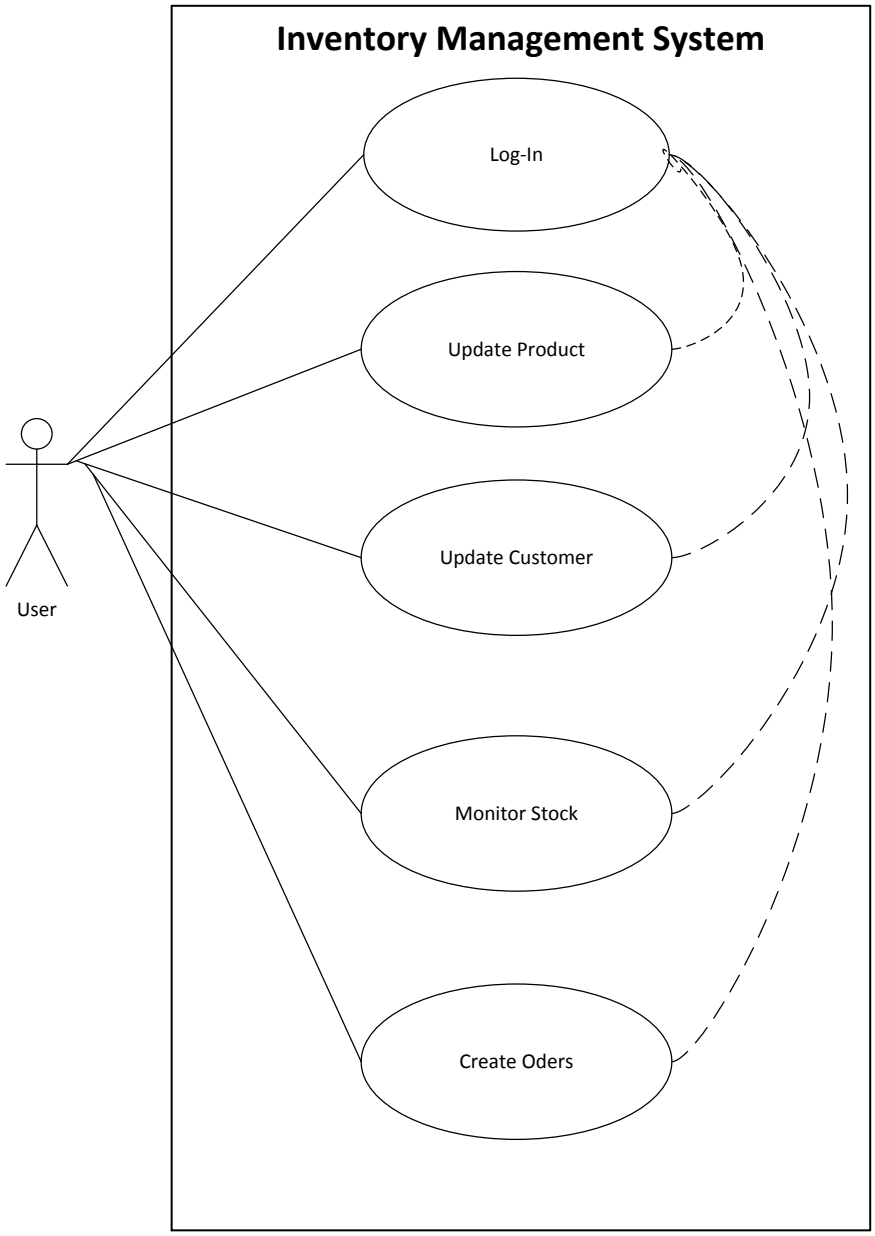


Figure 3-7 USE Case Diagram Inventory System

The above use case diagram represents the interactions between the user and the system, which is all the functionality that the user can perform within the inventory management system. In the diagram above, we can see that in order for a user to perform any inventory functionality, the user must be logged onto the system. Once the user is successfully logged onto the system, the user can update products and update customer information. This also includes creating the customer, monitor stock and create orders. The diagram is derived from the natural language requirements listed procurement module requirement table. From the use case

diagram, we can easily produce a UML class diagram.

Class Diagram

The below class diagram presents the main classes of the inventory management system i.e., Customer, Product, Order, User and item. The class diagram can be transformed to Z specification by the use of schemas.

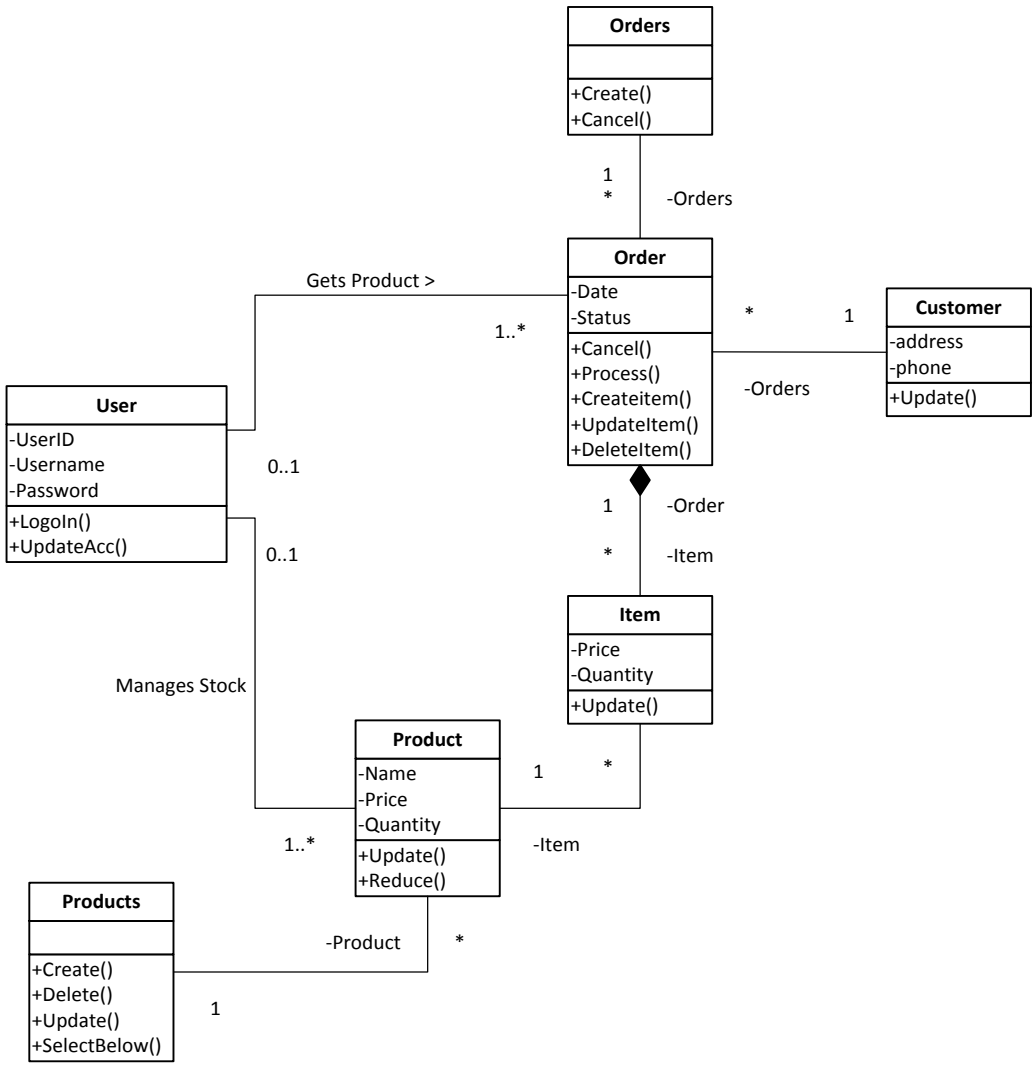


Figure 3-8 Class Diagram Inventory System (synthesised by the researcher)

Schach (2011) defines the class diagram as a method of determining entity classes and attributes relating to them. Furthermore, the class diagram defines the methods of the relating class and variables (Coates, 2012). This diagram is utilised mostly early, during the analysis stage of the software development process.

A formal specification in Z of the above system is developed next. For the purposes of this dissertation, some details are omitted from the specification, the reason being that its purpose is to illustrate the use of Z in developing a formal-methods adoption framework in the commercial world.

3.6.1 A Formal Specification

As mentioned previously, Z specification works with schemas when specifying requirements. Firstly, we will start by creating a schema for products. This specification will follow an established strategy for writing Z specification.

As per the (enhanced) Established Strategy for constructing a Z specification, basic (given) set are defined first.

Given Sets (basic Types)

From the requirements, the following basic sets are defined for the specification (basic types usually take singular denotations, e.g. USER instead of USERS).

[*STRING, AMOUNT, DATE*]
[*USER, PRODUCT, ORDER, CUSTOMER]*
STATUS: = pending | cancelled | processed

The above set of basic types will be augmented with feedback to the user of the system later in the specification (refer schema *ProductAlreadyExists* towards the end of this specification). The following terms are used in the specification and the below describes what each term stands for:

Next, a state space for the User entity is defined.

User

User

$$
\begin{array}{|l}
\hline
users : \mathbb{P}\ USER \\
userName : USER \pfun STRING \\
userPassword : USER \pfun VARCHAR \\
\hline
\mathrm{dom}\ userName = users \\
dom\ userPassword = users \\
\hline
\end{array}
$$

The above schema represents users that are maintained by the system. The schema further associates the password with the user. Schema *Log-in* towards the

end of this specification specifies an appropriate log-in operation for a user.

Product

Bellow schema defines products for the system.

Product

products: $\mathbb{P}$ *PRODUCT*
prodName: *PRODUCT* ⤔ *STRING*
prodPrice: *PRODUCT* ⇸ *AMOUNT*
prodQuantity: *PRODUCT* ⇸ $\mathbb{N}$
dom *prodName* = *products*
dom *prodPrice* = *products*
dom *prodQuantity* = *products*

Schema summary

Component *products* represent the set of characteristics of all the existing products in the system.

- *prodPrice*: *PRODUCT* ⇸ *AMOUNT* is a component of the state is declared partial function notation (⇸).
- *prodName*: *PRODUCT* ⤔*STRING: Since* the rules state that no two products can have the same name, a partial injective function is used to declare product names.
- The domains are specified in the predicate section of the schema. For this requirement, each attribute of the product should equal the identities collection. i.e.

  dom prodName = products
  dom prodPrice = products
  dom prodQuantity = products

Customer

Before creating an order schema, a customer schema must be specified. A customer is linked to an order, viz.

Customer

customers: ℙ *CUSTOMER*
custAddress: *CUSTOMER* ⇸ *STRING*
custPhone: *CUSTOMER* ⇸ *STRING*

---

dom *custAddress* = *customers*
dom *custPhone* = *customers*

Schema summary

- *customers*: ℙ *CUSTOMER represents* all existing customers in the system.
- *custA*ddress: CUSTOMER ⇸ STRING and *custPhone*: CUSTOMER ⇸STRING are partial function information about customers with respect to addresses and phone numbers.

Order

With existing products identified in the previous schema, there should be information maintained on existing orders in the system.

Order

order: ℙ *ORDER*
orderDate: *ORDR* ⇸*DATE*
orderStatus: *ORDR* ⇸*STATUS*
orderCustomer: *ORDR* ⇸*CUSTOMER*

---

dom *orderDate* = *order*
dom *orderStatus* = *order*
dom *orderCustomer* = *order*

Schema summary

- ℙ ORDERS represent components of all the existing products in the system.
- *orderDate: ORDR* ⇸ *DATE* is an attribute to the order with respect to the date of the order and it is declared using partial function notation (⇸).
- *orderStatus: ORDR* ⇸ *STATUS* is also an attribute which indicates the order status i.e., pending, processed, or cancelled.
- orderCustomer: ORDER ⇸CUSTOMER this represents the customer that has placed the orders. Each order must be linked to a customer.

The domains are specified in the predicate section of the schema. For this requirement, each attribute of the orders should equal the identities collection. i.e.,

dom *orderDate* = *order*

dom *orderStatus* = *order*

dom *orderCustomer* = *order*

3.6.2 Specifying Operations

Operations in Z represent dynamic aspects of the specification, this usually includes create, read, update, and delete operations.

Create Product Operation

A product is created through the following schema.

CreateProduct

$\Delta Product$
prduct?: *PRODUCT*
nme?: *STRING*
prce?: *AMOUNT*
qntity?: $\mathbb{N}$

$prduct? \notin products$
$products' = products \cup \{prduct?\}$
$prodName' = prodName \cup \{prduct? \mapsto nme?\}$
$prodPrice' = prodPrice \cup \{prduct? \mapsto prce?\}$
$prodQuantity' = prodQuantity \cup \{prduct? \mapsto qntity?\}$

When you observe the above schemas, the first necessity when creating operations is to declare the status of the state space, *Product* in this case. The schema states the state for *Product* is changed (or might change), owing to the operations specified, hence $\Delta Product$.

On the operations side, first is precondition stating a new product to be added is not in the database. The next predicates involving *products*, *prodNme*, *prodPrce* and *prodQntity* specify appropriate after states of the database components as indicated.

Create Order

The below schema specifies the placing of an order. Order status and date are specified and the person placing the order is captured.

CreateOrder

$\Delta Order$
date?: *DATE*
customer?: *CUSTOMER*
ordr!: *ORDER*

---

order! $\notin$ *orders*
ordr' = *orders* $\cup$ {*ordr*!}
orderDate' = *orderDate* $\cup$ {*ordr*! $\mapsto$ *date*?}
orderStatus' = *orderStatus* $\cup$ {*ordr*! $\mapsto$ *pending*}
orderCustomer' = *orderCustomer* {*ordr*! $\mapsto$ *customer*?}

Once the order has been created successfully, the next step is to process the order. The first part is to declare the order schema so as to ensure that create order and order schemas are linked. This also indicates that the state of the order may change, due to operations specified.

The first precondition states that this is a new order and it's not in the system. The next predicates involving *ordr!, orderDate*, *orderStatus* and *orderCustomer* maintain the date, status and customer information of the new order being placed. Note also that the system generates a new order number, hence the output symbol decoration of order, namely, *ordr!*

Create Customer

Naturally, new customers can also be added to the system. The *Customer* schema specified above serves to show all available customers in the system. It is also known as a static schema, similar to a UML class diagram.

The following schemas add a dynamic nature to the *Customer* specification and indicate their link via the Δ*Customer* notation. The first schema adds a new customer to the system.

CreateCustomer

$\Delta Customer$
customer?: *CUSTOMER*
address?: *STRING*
phone?: *STRING*

$customer? \notin customers$
$customers' = customers \cup \{customer?\}$
$custAddress' = custAddress \cup \{customer? \mapsto address?\}$
$custPhone' = custPhone \cup \{customer? \mapsto phone?\}$

On the operations side, first is the usual a precondition, stating that the customer about to be created must not be in the system. The next predicates specify after states for components *customers*, *custAddress* and *custPhone* as indicated.

Process Order

The following schema is for processing an order that was newly created or an order in the pending status.

ProcessOrder

$\Delta Order$
$\Delta Product$
$\Xi Customer$ (* *Yet, a real-life system would maintain some customer information* *)
product? : *PRODUCT*
ordr?: *ORDER*

$ordr? \in orders \land product? \in products$

(* *Valid pending order and product stock available* *)
$orderStatus(ordr?) = pending \land prodQuantity(product?) > 0$

(* *Components that remain invariant* *)
$orders' = orders$
$orderDate' = orderDate$
$orderCustomer' = orderCustomer$
$products' = products$
$prodName' = prodName$
$prodPrice' = prodPrice$

(* *New status of order* *)
$orderStatus' = orderStatus \oplus \{ordr? \mapsto processed\}$

(* *New quantity of product* *)
$prodQuantity' = prodQuantity \oplus$
$\{(product? \mapsto prodQuantity(product?)) - orderQuantity(ordr?)\}$

As per Z's schema inclusion, the process order schema includes three other schemas, namely, *Order*, *Product* and *Customer*. In addition, it also indicates that the states of *Order* and *Product* may change. For reasons of simplicity the state of Customer remains invariant – in a real-life system, some change in the customer-order relationship would be specified. The schema also input the new order identity (*ordr?: ORDER*), and *product? : PRODUCT*.

The schema validates that the order has been placed for a valid customer and the order status is pending when created. The schema specifies that before the value of the number of the specific product is positive, and that some components remain invariant as indicated. Should all predicates (precondition) hold, the after state of the status is specified accordingly and the product quantity on hand will be reduced by the quantity of the order.

Note also that standard Z has no notion of documentation of technical content in a

schema. Any documentation to a schema is stated as (English) prose in the discussion that follows a schema.

Update product

The next requirements on the order management module are that users should be able to update and delete products in the system. On the order side, users should be able to cancel the order that is not yet processed. The update, delete and cancel operations adjust the entity value on the system, either to be more (acquire) or less (sell-off or write-off).

The following schema specifies the update operation:

UpdateProduct

$\Delta Product$
$product? : PRODUCT$
$nme? : STRING$
$prce? : AMOUNT$
$qntity? : \mathbb{N}$

$product? \in products$
$prodName' = prodName \oplus \{product? \mapsto nme?\}$
$prodPrice' = prodPrice \oplus \{product? \mapsto prce?\}$

(* *Abstracting away from order-product relationship in schema ProcessOrder above* *)

$prodQuantity' = prodQuantityy \oplus \{product? \mapsto qntity?\}$

Because we are updating the product, we first start by inserting the product schema on the UpdateProduct schema. Then, the first predicate validates that the product to be updated must exist in the system. Consequently, there is a remapping of *prodName*, *prodPrice* and *prodQuantity* functions to link them with the new name, price, and quantity values respectively of the existing product. The relevant Z operation for this purpose is denoted by the $\oplus$ relational override notation.

Delete product

This operation removes a product from the system.

DeleteProduct

$\Delta Product$
product?: *PRODUCT*

---

product? $\in$ *products*
products' = *products* $\setminus$ {*product*?}
prodName' = {*product*?} $\lhd$ *prodName*
prodPrice' = {*product*?} $\lhd$ *prodPrice*
prodQuantity' = {*product*?} $\lhd$ *prodQuantity*

The *Product* schema is included in *DeleteProduct* schema and indicates a possible state change. The precondition *product?* $\in$ *products* state that the product that is going to be deleted exists in the database (an appropriate error condition could be generated otherwise).

The specification indicates appropriate after states of components *prodName*, *prodPrice* and *prodQuantity*. This is archived by eradicating the state of the product that is about to be deleted (*product*?). The predicates are denoted by the symbol, $\lhd$ called domain subtraction.

Cancel Order

Referring back to the natural-language requirement stated earlier, the order can be cancelled only if it is still in the pending status. This can be archived by the following schema.

CancelOrder

$\Delta Order'$
ordr?: *ORDER*

---

ordr? $\in$ *orders*
orderStatus(*ordr*?) = *pending*
orderDate' = *orderDate*
orderStatus' = *orderStatus* $\oplus$ {*ordr*? $\mapsto$ *cancelled*}
orderCustomer' = *orderCustomer*

Firstly, the order has to exist in the system for it to be cancelled. The next predicate checks the status of the order to be cancelled. The order is subsequently cancelled. For the purposes of this specification, the *orderDate* and the customer involved in the order are not affected.

Enquiry Operation

As an illustration, we specify a simple enquiry on the system. Only one report will be specified for this dissertation.

SelectProductsBelowThreshold

$\Xi Product$
$quantity?: \mathbb{N}$
$products!: \mathbb{P}\ PRODUCT$

$products! = \{p: products \mid prodQuantity(p) < quantity?\}$

The above schema selects all the products that are below a certain threshold, hence any orders for these would be affected. By adding more rules in the schema, users can be notified once the product is below the specified threshold.

The above schema will select all the products that are below-set threshold.

Total Operation

Lastly, total operations that cater for partial (correct) operations above, as well as cases where the preconditions do not hold can be specified for the order system.

To cater for total operations, one has to define success schemas, as well as schemas for error conditions. I give one example of these below.

Operation success

Success

$result!: REPORT$

$result! = success$

The above schema presents the results of successful inputs when creating a product. If the precondition holds, the final outcome will be successful.

Below is an error schema for an attempt at creating a product that already exists.

Error condition

ProductAlreadyExists

Ξ*Product*
prduct?: *PRODUCT*
result!: *REPORT*

---

prduct? $\in$ *products*
result!: = *product_already_exists*

Once error conditions are added, the specifier should augment the data type definitions given earlier as follows (basic type *REPORT* added):

[*STRING, AMOUNT, DATE*]
[*USER, PRODUCT, ORDER, CUSTOMER, REPORT]*
STATUS: = pending | cancelled | processed

Using Z's schema calculus, a robust operation *RobustCreateProduct* for creating a product can be specified:

$$RobustCreateProduct \mathrel{\widehat{=}} (CreateProduct \wedge Success) \vee ProductAlreadyExists$$

In expanded form, *RobustCreateProduct* will check for both success and error conditions.

RobustCreateProduct

$\Delta Product$
product?: *PRODUCT*
name?: *STRING*
price?: *AMOUNT*
quantity?: $\mathbb{N}$
result! : *REPORT*

(*product*? $\notin$ *products* $\wedge$
products' = *products* $\cup$ {*product*?} $\wedge$
prodName' = *prodName* $\cup$ {*product*? $\mapsto$ *name*?} $\wedge$
prodPrice' = *prodPrice* $\cup$ {*product*? $\mapsto$ *price*?} $\wedge$
prodQuantity' = *prodQuantity* $\cup$ {*product*? $\mapsto$ *quantity*?}) $\wedge$
result! = *success*)
$\vee$
(*product*? $\in$ *products* $\wedge$
products' = *products* $\wedge$
prodName' = *prodName* $\wedge$
prodPrice' = *prodPrice* $\wedge$
prodQuantity' = *prodQuantity* $\wedge$
result! = *product_already_exists*)

As indicated at the beginning of the specification an operation to log a user onto the system can be defined:

Log-in

Log-in

$\Delta User$
username?, *password*?: *User*
r! : *RESULT*

If username? $\mapsto$ *password*? $\in$ *User*
r! = *Success*
Else
r! = *Failed*

The above schema represents the login requirement (authentication). It checks whether the captured username and the password (both input to the specification) pair matches, and if they match, the user will be logged in successfully onto the system and perform stock management functions as specified earlier. If the username and password pair do not match, the result is a fail and the user cannot perform any functionality within the system.

Note that although Z is an abstract specification language, it allows for procedural constructs like “If” and “Else”.

System

The full state of the procurement system can be defined through schema inclusion as indicated below. For this dissertation, the full Z specification was not written for this system. Potter, Sinclair and Till (1990) state that it is convenient to draw a schema demonstrating the whole system state. Therefore, the system state of this system is:

System

User
Product
Order
Customer

Discussion of Specification

A Z specification was created for the order management part of the procurement module. The first schemas are the static schemas, which present the users, products, customers and the orders that are to be maintained for the system.

The next part of the specification defined the operations to be performed on the above static components. These are to create and maintain customers, products and orders; and allow a user to log onto the system. Various operation schemas for these were defined above. The operations defined were partial, in the cases at hand where the precondition was satisfied in each case, i.e. for a correct, intended version of the operation. Following that, one example each of success and an error schema was defined and it was indicated how these could be combined to define a robust operation, namely, *RobustCreateProduct*. Lastly, a schema (*System* above) was defined that includes the static specifications of the entities.

Following the specification above the last proposition can be defined.

> ***PROP 5:*** *Using the Z notation as an entry language ought to facilitate the adoption of formal methods. Z is believed to be easy to learn and apply. Only basic mathematical set theory and 1st-order logic are required.*

Following the preceding discussions in this chapter the following, preliminary formal-methods adoption framework emerges.

3.7 Preliminary Framework

Quickening the Adoption of FM in Business

The literature review and the propositions made throughout the dissertation lead to the development of the Preliminary Framework. The researcher has categorised ways that may assist in quickening the adoption of formal methods under four headings namely education, remuneration, open-source and support tools. Each element is explained further below.

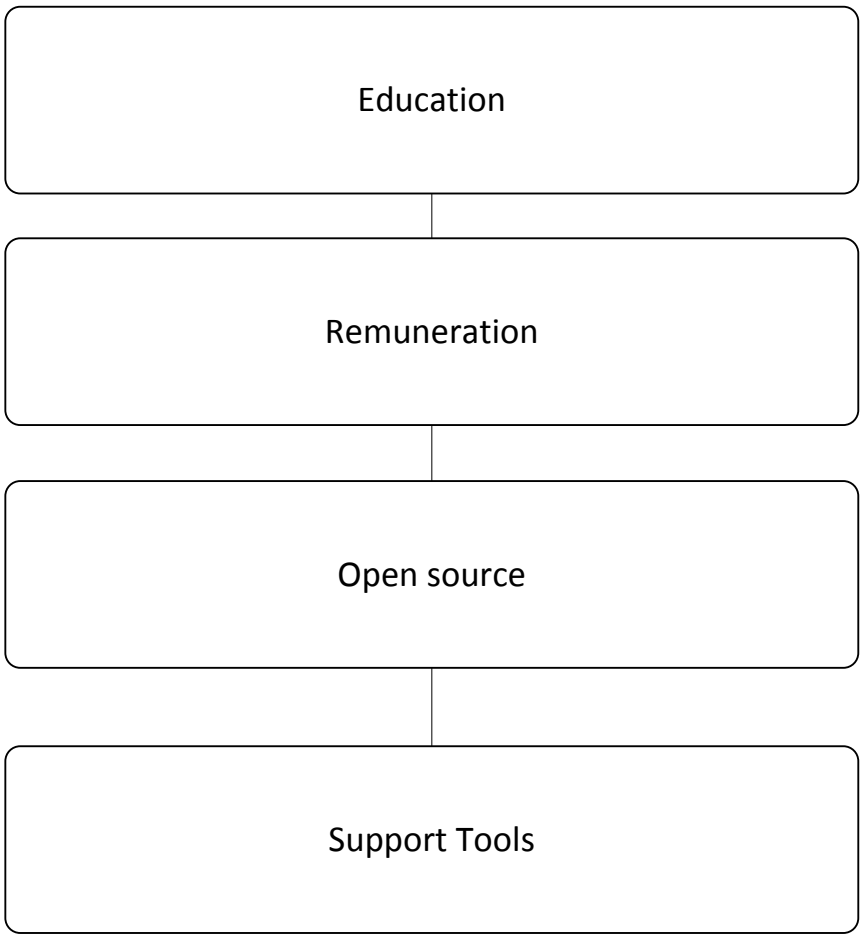


Figure 3-9 Preliminary Framework

Education

Arguably the most prominent issue in the adoption of any technology is knowledge on the technology, in this case, FMs. Baier and Katoen (2010) state that “FMs should be part of the education of every computer scientist and software engineer, just as the appropriate branch of applied maths is a necessary part of the education of all other engineers.” Parnas (2010) has suggested that students ought to be taught only detailed programming, rather than any other factors of the software development process, such as methods and design principles. The researcher views this statement as treacherous, as it ignores other aspects of the software development process and gives the impression that the success of a software project is achieved by coding and debugging.

To facilitate the adoption of formal methods in the commercial world, formal methods ought to be introduced as part of the syllabus as early as high school computer education. The introduction of formal methods into high school and undergraduate education ensures that the new generation of engineers are aware of it, and hopefully know how to use it. Books need to be written about formal methods, and these writings ought to be easily accessible. Proper FM certification needs to established and awarded to individuals who qualify to use formal methods.

Remuneration

Good compensation to FM specialists will also attract more people to join the formal methods field. It should also motivate students and people already in the computer science industry to obtain certification. Established professional societies (e.g. the IEEE) can provide standardised teachings in the use and practical application of formal methods. To get the necessary buy-in from engineers, training or awareness of formal methods should be provided as a top-down approach i.e., from top management to senior managers, then analysts, and then developers. Almost all the companies in today’s world are concerned with cutting cost, and that’s what top management understands as a priority. Hard evidence of cost-saving, reduction in development time, and the improved quality of the resultant system when using FMs, ought to be made available to the portfolio and programme managers.

Open-source

Encouragement for the use of FM on open source software can fasten the adoption in business. Internet communities or blogs should be formed, where people can discuss issues, challenges and success stories about the use of formal methods.

Software users need to have some knowledge about formal methods, for example, how and when to apply FMs.

The benefits of formal methods need to be presented and made public. This can be achieved via the use of weekly newsletters or research websites such as Gartner, publishing stories of companies that have successfully developed and implemented systems using formal methods. In South Africa, we have an institute called the CSIR (Council for Scientific and Industrial Research), which is a leading technology research organisation in the country. This institute can be used to promote formal methods.

Support tools

FMs tools should be user-friendly, and ought to allow for the easy integration into current development environments such as the .NET framework and Linux development frameworks. It should also be possible to integrate FM artefacts and techniques with existing SDLCs. As more and more computer devices are mobile e.g., smartphones and tablets, more apps for these devices should utilise some form of formalisation. FM tools should have more automation, and also automated test generation. Lastly, formal methods ought to facilitate clear estimations as to how long it will take to perform analyses. Van der Poll (2010, p. 48) states that “it must be possible to regularly measure the progress made by formal technique during development”, hence dynamic measurements ought to be possible.

3.8 Summary

This chapter discussed formal methods using an application from the ERP system domain. An explanation was presented on what ERP systems are and the types of modules found in ERPs i.e., procurement, human resource, finance, etc. Each module was briefly discussed. A graphical comparison of Software development costs when using informal- or semi-formal methods vs using formal methods was presented and explained.

A short case study was specified in UML and the class diagram was transformed into the state spaces of a number of entities, followed by some operations (schemas) on these. An informal specification was written based on the HR module, requirements listed in a natural language, then a process was mapped on the employee leave application. A use case diagram was developed, and the details around the use case explained.

Before specifying a formal specification, basic mathematical set theory was discussed and a high-level explanation of a Venn diagram was given. For the formal specification, an ordering (procurement) system was chosen. Schemas were specified in terms of a number of state spaces; and updating, read, and deletion schemas, following the enhanced established strategy for constructing a Z specification. A summary of the formal specification was given. Lastly, to close off the chapter, a preliminary framework was presented aimed at fastening the adoption of formal methods in the commercial world.

From Chapter 3 we can conclude that using formal methods in terms of the Z notation can assist in clearly specifying requirements. Teaching Z as a formal method entry language can assist in the uptake. Some background in discrete mathematics would assist in learning formal methods.

The next chapter will look at the research methodology followed in this dissertation. This is explained by using Saunders et al.'s (2015) research onion. Data collection methods used in the research will be discussed.

Chapter 4 Research Methodology

4.1 Chapter Layout

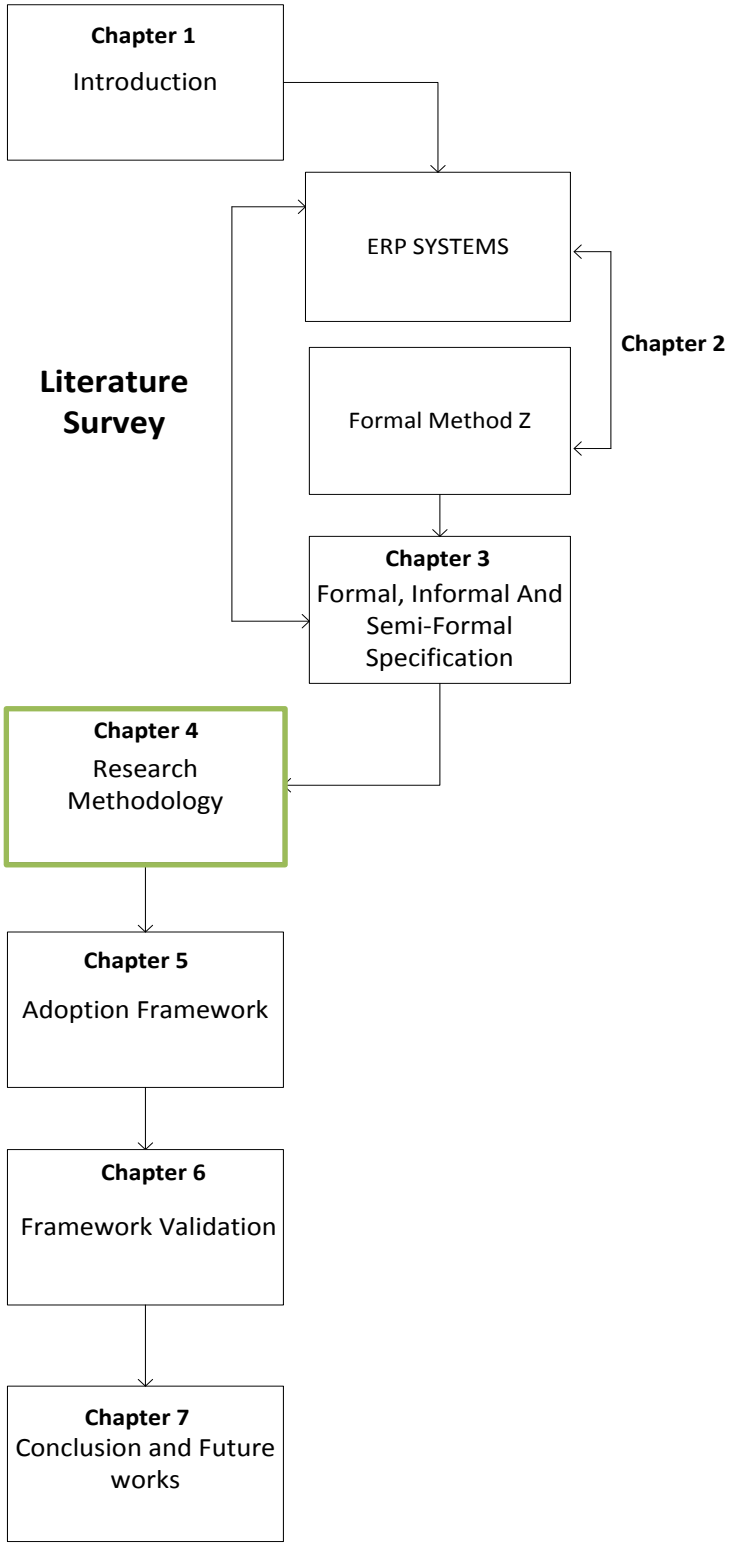


Figure 4-1 Chapter Layout

4.2 Introduction

The previous chapter discussed formal methods in the ERPs system. It also described a brief case study on the ERP system, where informal requirements were elicited from the case study and transformed into formal requirements. The last chapter started by explaining what ERP systems are, and the types of modules in ERP systems, namely, a procurement module, a human resource module, a finance module, and the contract management module. Each module was briefly discussed. A graphical comparison of Software development costs when using informal methods vs. using formal methods was presented and explained. The informal specification was then written based on the HR module, requirements listed in a natural language, and a process mapped on employee leave application. A use case diagram was developed, and the details around the use case explained. Before specifying a Z specification, set theory fundamentals were discussed. Lastly, the essentials of a Z specification on the procurement module was specified.

Chapter 4 presents and discusses the rationale for the chosen research methodology. The chapter will make use of the research Onion diagram developed by Saunders et al (2015). Each item will be explained in the diagram, in terms of how it relates to this dissertation. Furthermore, this chapter will expand deeper into the methodology as well as data collection methods used to gather relevant information in order to contribute to this field.

4.3 Research Onion

Before I present the Research Onion, I would like to give a brief definition of the operational concept of research used here. One of the reasons why research can be conducted is to obtain new knowledge or information and to contribute new findings to the body of knowledge (Oates, 2006). Rajasekar et. al. (2016b) further listed what research can enable one to archive:

- finding new facts;
- devising a solution to either scientific, technological and social problems;
- verifying those findings; and
- lastly, developing theories, tools and ideas to solve present problems.

Next is the presentation of the Research Onion Diagram. This diagram is adopted from Saunders et al. (2015):

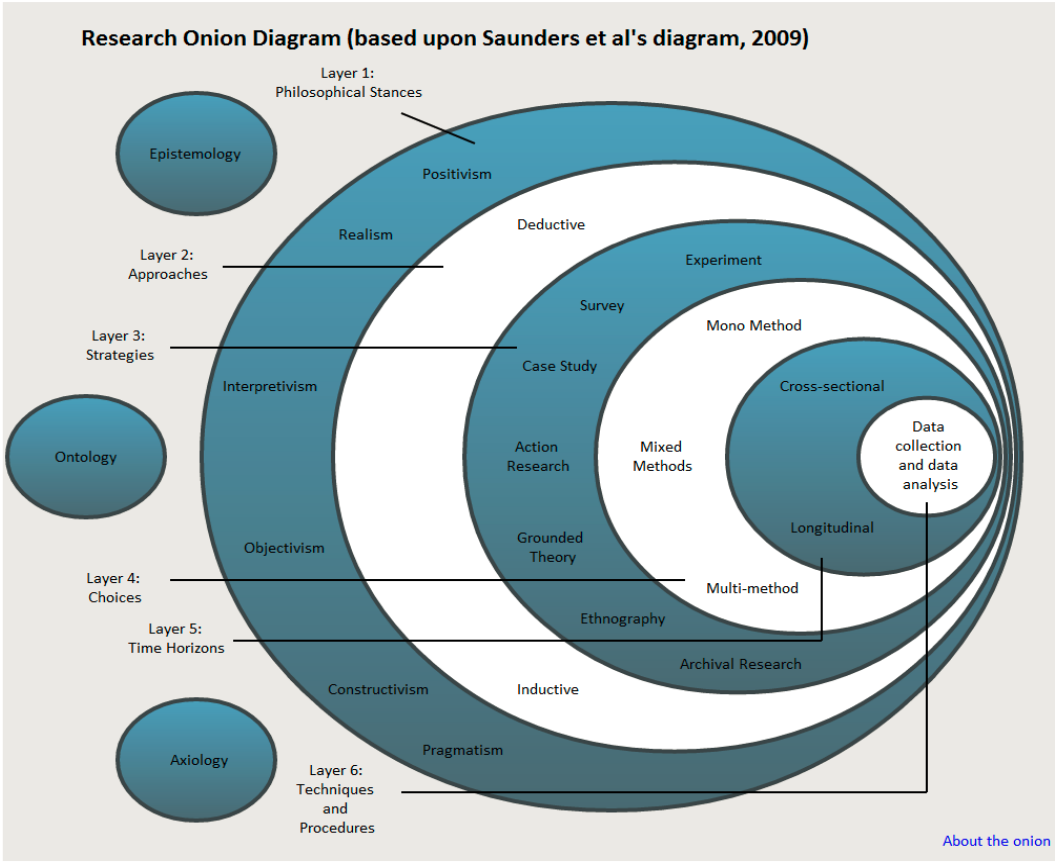


Figure 4-2 Onion Diagram (Saunders et al., 2015)

Saunders et al.'s (2015) research onion (Figure 4-2, synthesised from their 2009 version) is utilised by analysis starting from the outer layers, to the inner layers. At each layer, the researcher explains whether it is applicable to the research or not.

The onion discusses the three philosophical categories of epistemology, ontology, and axiology. These philosophies are important, as they guide the researcher in planning and conducting the research itself.

Epistemology: mostly used in scientific research as it focuses on facts and information that can be proved without any contestation or influences of the situation and someone's opinions. The researcher aims at distinguishing true knowledge from factual knowledge, as achieved by arduous testing (Norris, 2005). Epistemology is often thought of "knowing what it is that you know".

Ontology: how society view reality, as opposed to reality itself, and how our views influence people's behaviours. It also assists in knowing how society influences our settings (Saunders and Lewis, 2015). An ontology is often viewed as a set of concepts and categories in a domain (e.g. a subject area), indicating their properties

and relationships among them.

Axiology: values and ethics. This philosophy allows researchers to comprehend how their views and values influence the collection and analysis of the research. This research will be guided by the axiology philosophy, as it is mostly theoretical in nature. Even though we are considering mathematical notation, no actual experiments will be conducted to prove or reject a hypothesis.

Layer 1: Philosophical stances

The Onion is divided into six layers. The outer layer, which is layer one, is called the philosophical stance. The philosophical stance will guide the researcher as to how to ultimately gather and analyse data in order to formulate relevant findings. The first layer encompasses objectivism, which distinguishes that social occurrences, and their connotations, which occur separately to that of social actors. Constructivism is the reverse of objectivism, where social actors create social phenomena. Positivism is a philosophical paradigm with two assumptions i.e., our world is ordered and regular, not random, and we can investigate it objectively. Gieryn and Giddens (2006) state that positivism makes the assumption that reality is known, and is also focused on finding facts, using techniques, and gaining knowledge to solve a problem.

Realism is similar to positivism, in the sense that its processes and belief that social reality and the researcher are independent of one another, and so will not create biased results. Interpretivism speaks to methods highlighting the meaningful nature of people's involvement in cultural and social life. Oates (1998) defines interpretive research in Computing or Information Systems as the way of understanding the social context of information systems i.e., the influence that the social setting has on the development of information systems by people. According to pragmatism, both objectivism and constructivism are the correct ways to conduct research.

This research is guided by positivism philosophical stance as we try to find how to increase the adoption rate of formal methods in the commercial world. This is the study of people concerning how they can easily adopt a new way of work. Also, the study and use of FMs embody mathematical aspects reminiscent of positivism.

Layer 2: Approaches

Layer 2 approaches have two components, namely the deductive and the inductive.

The difference between deductive and inductive reasoning is that in deductive reasoning, before starting the research, the researcher begins with a question or a statement that the researcher aims to answer, and in so doing aims to prove a theory, or validate a framework, model, etc. Inductive reasoning refers to when there is little to no research that exists on a given topic, where the researcher is aiming to create own theory or develop a framework, model, etc. (Smith, 2017). This research has a mix of both, where it is initially inductive, as the researcher will build a framework to address the slow adoption of formal methods. This research is also deductive in nature, as it tries to answer the question as to why there is slow adoption of formal methods in the commercial world, and how can we increase the adoption of formal methods. In essence, therefore, the researcher will validate the framework developed in this work.

Layer 3: Strategies

Layer 3 refers to the research strategies, that is, the methods that will be used to collect and analyse data for the research. This can be an experiment, survey, case studies, action research, grounded theory, ethnography, and archival research. Most of the research in this dissertation was done using the journals of the work that has already on this topic. Case studies on the companies that have successfully implemented systems using formal methods have been considered in Chapter 2.

In detail, the following strategy was used:

- *Documents* of work that has been already done on formal methods were collected and studied. Documents pertaining to the myths and the different types of formal methods were used as input to this dissertation;
- *Case studies* relating to formal methods were scrutinised and conclusions drawn from them. Case studies of companies using formal methods in their software projects were also used as input to this research; and
- *Internet (scholar's sites)* was used most of the time to gather the documents and e-journals. Mainly scholarly sources were used.

Layer 4: Choices

At this layer, a researcher decides whether to use quantitative or qualitative research or both. The layer encompasses the following: A mono-method research refers to when one of the data collection methods is used, which can be quantitative

or qualitative. Mixed methods research refers to using both quantitative and qualitative research methods. Lastly, multi-methods refers to when the researcher chooses to use quantitative data and qualitative data, but the researcher's viewpoint is embedded in one or the other (Andrew and Halcomb, 2009).

Multi-methods apply to this research, where most parts are qualitative, studying documents, cases studies, and internet journals. A modicum of quantitative research was done when considering the discrete mathematics and logic aspects of FMs. (Note that while quantitative work usually involves just (real) numbers, the researcher views specifics embedded in FMs as being quantitative) comparing the costs of using formal methods vs the costs of using semi- or informal methods. Most of the focus was placed on qualitative research. According to Hamilton-Smith (2001), a goal is to deliver answers to questions that are asked frequently in a given research scenario, by using already-defined steps to obtain the answers to the questions. Qualitative research pursues hard evidence and provides new findings to the research and the body of knowledge. It should be noted that one of the shortcomings of a qualitative methodology is that it may lack a generalisation of the findings (Oates, 2006).

Layer 5: Time Horizons

Generally, there are only two-time horizons, viz. cross-sectional and longitudinal. Both time horizons can use qualitative or quantitative research or both where the difference is that cross-sectional research is for the shorter term, or short period of time and the longitudinal is for the longer term. Adoption of formal methods in the commercial world research is to be completed in the medium term, which opted the researcher to use a cross-sectional time horizon.

Layer 6: Techniques and procedures

This is usually the final aspect of the research to consider, in which the researcher needs to make sense of all the data collected and makes a decision as to what works best and what doesn't work. All the decisions made at this stage must be adequate, with all the layers that are philosophies, philosophical stances, strategies, choices, and time-horizons. Analyses and conclusions following the data collection will detail the outcomes of the research.

The research process followed in this dissertation is depicted in Figure 4-3 below and discussed thereafter.

4.4 Research Process Diagram

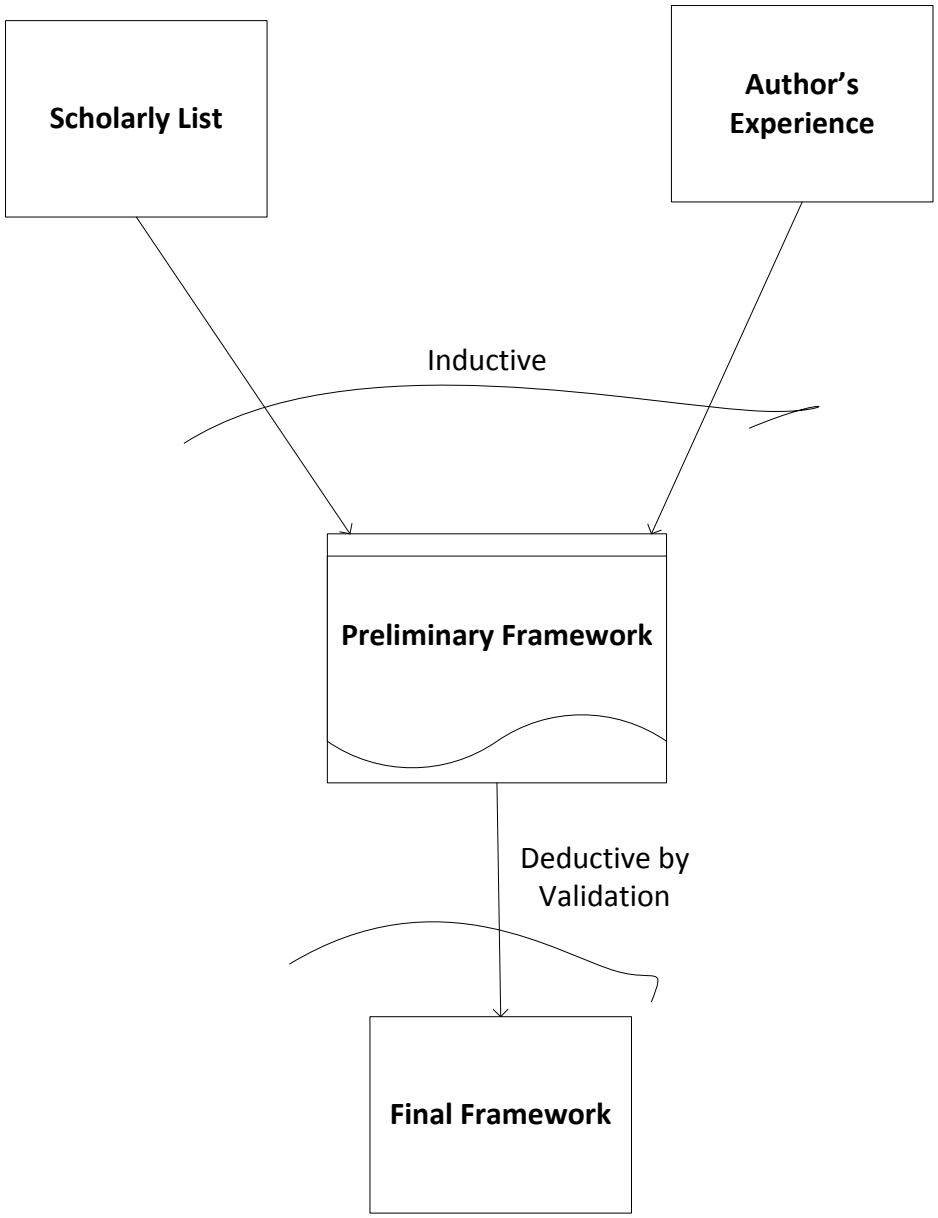


Figure 4-3 Research Diagram (Synthesised by the researcher)

Scholarly List

At first, the researcher looked at the current documentation of formal methods. The documentation as mentioned above are comprised of online journals, printed books, and online books, and the PhD theses and MSc dissertations written by other students on this topic. References and acknowledgements have been cited for documentation used. Information on the ERP system introduced earlier is based mostly on scholarly papers.

Author's Experience

The author's experience was also taken into consideration when conducting this research. The author works as a business analyst with about six years' experience in the IT field, exposed to telecommunication, insurance, and software-specific industries, involving requirements elicitation, process engineering, optimisation, testing and product development. Throughout my six years of experience, I haven't encountered a company using formal methods, or even used them in my workplace.

Learning about formal methods helped me to garner another view of the software development process. My experience has influenced the development of the framework on the adoption of formal methods in the commercial world.

Scholarly lists and the author's experience is *inductive* in nature. Inductive refers to when there is little to no research that exists on a topic, where the researcher is aiming to create their own theory. Developing one's own preliminary framework is inductive, that is, part of the layer 2 approaches on the onion diagram (refer above).

Preliminary framework

A literature review along with industry experience served as an input to the preliminary framework. As indicated, the preliminary framework is comprised of education, which refers to educating software engineers/specialists about formal methods at the early stages, where remuneration also forms part of the framework, meaning that if software engineers are encouraged to use formal methods, they ought to receive high(er) remuneration. The open-source software that has been developed using formal methods will also make formal methods fashionable. Lastly, in terms of the preliminary framework, support tools that improve on the user experience (UX), i.e. easy to use and understand should be developed and made available.

Deductive, for this research, aims to validate/justify and further enhance the preliminary framework aimed at fastening the adoption of formal methods in the commercial world. An improved formal-methods adoption framework will be presented in the next chapter.

Final framework

The final framework was then produced, incorporating all the tasks on the

methodology i.e., scholarly works, author's experience, and the preliminary framework. The final framework has some additional items, e.g. buy-in from top-management within the companies. Publications of formal methods ought to be made available via blogs, newsletters, the world wide web, etc. and lastly, the results on the use of formal methods ought to be made available, whether it is a positive or a negative result. Qualitative propositions were stated in earlier chapters, assisting in the drawing up of the final framework which is presented in the next chapter.

The final framework is presented in Chapter 5 which follows next, and it is subsequently validated through a case study in Chapter 6. All these assisted in placing the formal methods adoption framework into a practical context.

4.5 Summary

Chapter 4 focused on the research philosophies followed in this dissertation, and the motivation as to why they were followed. The research onion diagram developed by Saunders et al. (2015) was used as the main reference. Each element on the diagram was explained in terms of how it relates to this dissertation, if not, then why. Furthermore, the chapter expanded deeper into the methods used to gather relevant information for this study. A research diagram was also presented.

The following chapter will develop a proposed framework aimed at increasing the adoption and use of FMs in the commercial domain. The proposed framework will be based on the findings of this dissertation and the work that has been done in this field by other researchers. The framework is presented in a tabular format, along with the diagram.

Chapter 5 FMs Adoption Framework

5.1 Chapter Layout

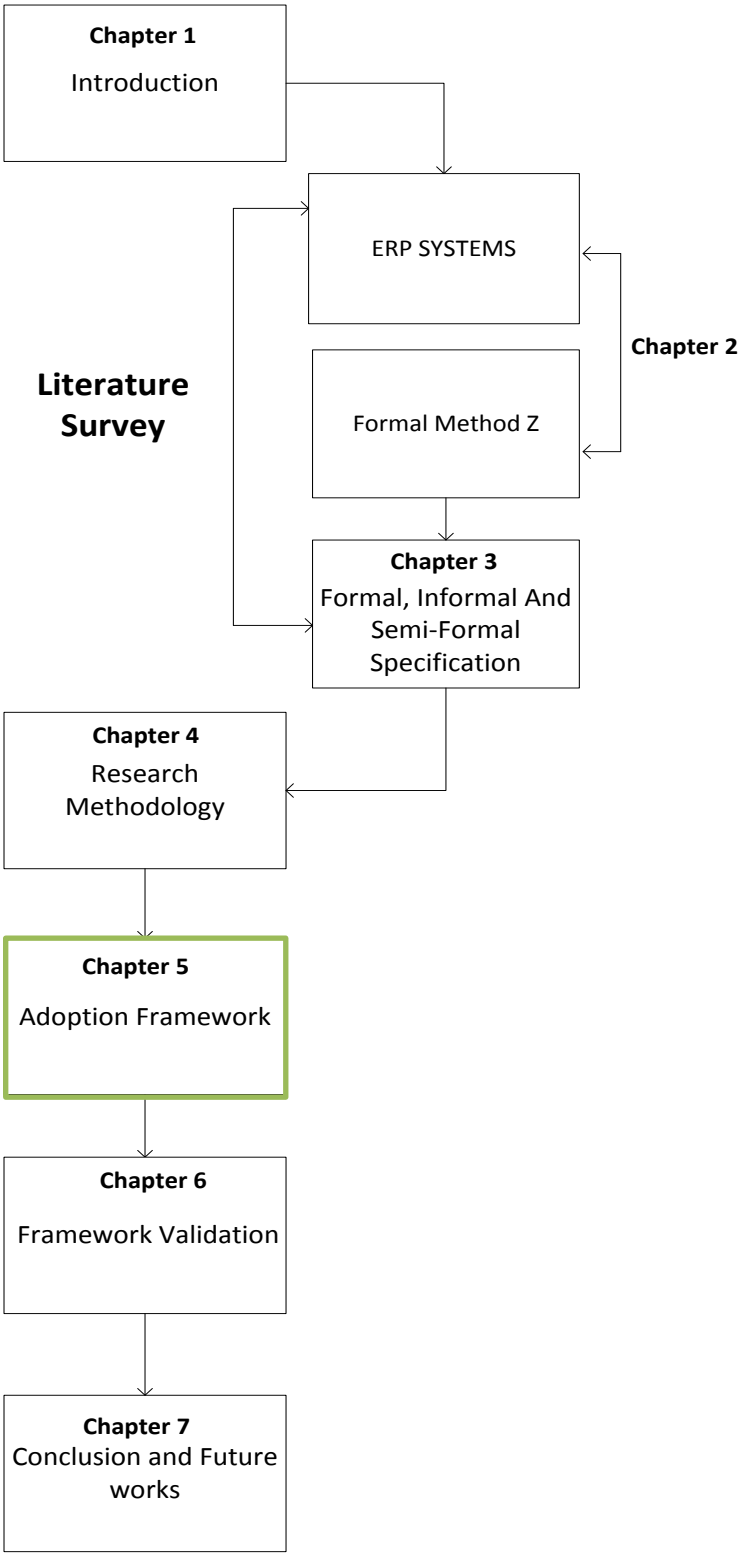


Figure 5-1 Chapter Layout

5.2 Introduction

The previous chapter focused on the research philosophies followed in this dissertation and the motives for choosing a certain methodology. The previous chapter also made use of the research onion diagram developed by Saunders et al. (2015). Each aspect of the diagram was explained, particularly in terms of how it relates to this dissertation. Furthermore, the previous chapter details the methodology, as well as data collection methods used to gather relevant information. Lastly, the validity and reliability of the project were noted.

This chapter proposes a framework that the researcher hopes will aid in increasing the adoption of formal methods in the commercial world. The proposed framework will be based on the findings of this dissertation and the work that has been done on this field by other researchers.

5.3 Adoption Framework

A framework can be defined as a skeleton or a basic structure of the underlying system (Egon and Robert, 2003). This can be edited as required by deleting or adding items. From a software perspective, it can be defined as a set of functions within a system, and how they interconnect.

The following is the list of propositions formulated throughout the dissertation in earlier chapters. The propositions led to the development of the Formal Methods Adoption Framework:

> ***Proposition (Prop) 1:*** *Education plays a major role in formal methods adoption. This includes educating from the high school level to the university level as well as organisational training in the use of formal methods. Such education plays a pivotal role in the adoption framework.*
>
> ***Prop 1.1:*** *In addition to the above proposition, formal certificates and diplomas in formal methods ought to be created and awarded to those who qualify. Certification authorities should be well informed about the benefits of formal methods.*
>
> ***Prop 2:*** *Buy-in from all the business stakeholders is necessary for FM adoption. Getting Top-level management to agree to and accept the use of*

formal methods may well result in the whole organisation adopting formal methods.

PROP 3: *Widely accepted principles and guidelines on FMs can improve the adoption thereof. Practical, real-world examples of FMs successes and failures must be published in the software engineering and management communities.*

PROP 3.1: *Widely accepted principles and guidelines on FMs can improve the adoption thereof. Practical, real-world examples of FMs successes and failures must be published in the software engineering and management communities. Publications of formal methods successes in terms of cost savings in projects, clear specifications produced, and the overall final product delivered with fewer defects will raise much interest needed for the adoption of FMs. Practical, real-world examples of FMs successes and failures must be published in the software engineering and management communities.*

PROP 4: *Tools that are readily available and up to date with the latest technology should facilitate the adoption of FMs. Such tools ought to be integrated with the requirements management software and standard software programming tools e.g., MS Visual Studio.*

PROP 5: *Using the Z notation as an entry language ought to facilitate the adoption of formal methods. Z is believed to be easy to learn and apply. Only basic mathematical set theory and 1st-order logic are required.*

Table 5-1 summarises the components of the proposed FMs adoption framework:

Element	Description
EDUCATION	Software engineering education in the early stage
	Introduction to formal methods for the first-year university students
	Universal formal methods standards
	University accreditation specifically on formal methods
	Set theory basics at an early stage of educations systems
	Step-by-step guide on transforming informal requirement to the formal specification
	Knowledge sharing and common terminology
BUY-IN	Public sector using formal methods for their systems
	Enterprise top management buy-in
	Project manager and senior manager buy-in
	Training companies
	IT community buy-in
	Formal method language e.g. Z
REMUNERATION	FM specialist salaries, scare skill
ENVIRONMENT	IT environments where FMs are going to be utilised
	Tools to write a formal specification
	Integration of MS office to formal specification languages
	Open-source tools
	A collaborative environment for formal methods specialist to meet
	Built the right attitude within teams, team buildings
SUPPORT TOOLS	LaTeX, Alloy, Rodin/Event-B tool

Element	Description
PUBLICATIONS	Successful use of formal methods should be published daily, or as often as is feasible
	Forums i.e., internet new letters of formal methods
	Encourage the use of formal methods on open source systems
	Library catalogue on formal methods
RESULTS	Positive and negative results should be made available
	Description of each successful components of the system built using formal methods
	System developed using formal methods used in the real business environment
	Positive and negative results should be made available

Table 5-1 Formal Methods Adoption Framework

5.4 Adoption Framework Diagram

The formal-methods adoption framework proposed in this dissertation is given in Figure 5-2. It is envisaged that all the steps can be performed in parallel. The larger boxes (EDUCATION, PUBLICATION, RESULTS) indicate that the more we focus on that step the higher the probability that the adoption will be a success.

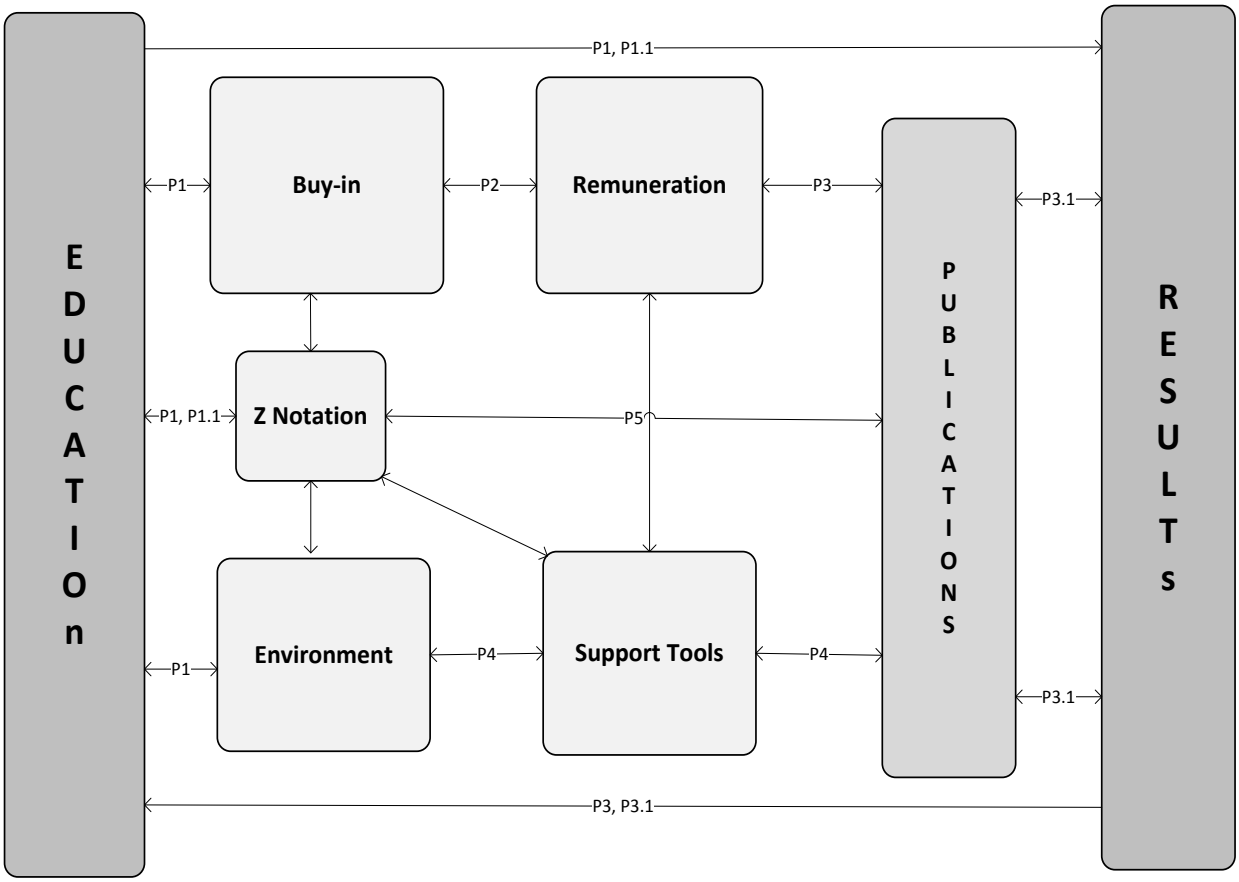


Figure 5-2 Adoption Framework Diagram

Each component of the framework is discussed below. Components will be discussed in the following order: EDUCATION, BUY-IN, REMUNERATION, ENVIRONMENTS, SUPPORT TOOLS, PUBLICATION, and RESULTS. Each component makes reference to the Z notation; therefore, Z will not be discussed separately. Examples of Z specification documents appear in earlier chapters.

EDUCATION: forms a foundation to allow for the adoption of FMs in the commercial world. Without proper education and transfer of skills to the new upcoming engineers, the current state of formal methods would, arguably, stagnate. As observed in Table 5-1 above, education should take place from an early level, as far

back as high school, where a basic introduction is made to formal methods and then to the university level where first-year computer science and software engineering students will be learning about formal methods as a module by itself. Education will also cover the challenge of not having common terminology in formal methods. People who qualify or pass this course ought to be recognised by awarding proper accreditation. The Z notation is relatively easy to teach and learn, as it requires basic mathematical knowledge. Students and IT professionals might start by learning Z, and they can peruse other formal specification languages when they get more conversant with the FMs arena.

BUY-IN: simply refers to people embracing the idea of using formal methods within their organisations. Buy-in is needed from top management right through to the project manager, and the engineers. Public enterprises buy-in is also important to utilise formal methods in the development of their systems. Furthermore, buy-in from companies that provide IT training courses and the IT community in general, will have a huge impact on formal-methods adoption. Established professional societies (e.g. the IEEE) can provide standardised FM teachings in the use and practical application of formal methods. To secure the necessary buy-in from engineers, training or awareness of formal methods should be provided following a top-down approach i.e., from top management to senior managers, followed by analysts, and then developers. With Z as a recommended formal-method language for this research, top management needs to understand the benefits that Z brings, where it is taken as easy to comprehend, and also flexible enough to model a specification that leads to correct code, and so forth. Because most people are familiar with Z syntax and semantics, Z is the most-used formal specification language (Bowen, 2016).

REMUNERATION: Attractive compensation for FM specialists will also attract more professionals to join the formal methods field. It will motivate students and people already in the computer science industry to gain certification. Almost all the companies in today's world are concerned with cutting costs, which top management understand as a priority. Hard evidence of cost-saving, reduction in development time, and the improved quality of the resultant system when using FMs, ought to be made available to the portfolio and programme managers.

ENVIRONMENT: where formal methods are used. The environment encompasses the proper tools to utilise formal methods. Current software engineering tools ought to be integrated with formal-methods tools to allow a smooth transition to FMs usage.

Teams working on formal methods must have the right mindsets and attitudes, which can also be influenced by the environment in which they work. Encouragement of the use of formal methods in open-source software will bring about new ideas and lead to a positive perception of formal methods among practitioners and managers (refer buy-in of management above).

SUPPORT TOOLS: FMs tools ought to be user-friendly and should allow for easy integration into the current development environment, such as the .NET framework and Linux development frameworks. It should also be possible to integrate FM artefacts and techniques to an existing SDLC. As more and more computer devices are mobile e.g., smartphones and tablets, more apps for these devices should utilise some form of formalisation. FM tools should have more automation, and also automated test generation. Lastly, formal methods should facilitate clear estimations regarding how long it will take to perform analyses. Van der Poll (2010) states that "it must be possible to continuously measure the progress archived by formal technique during software development".

PUBLICATION: Accounts of the successful use of formal methods should be published regularly. This can be made available via internet newsletters, fora, blogs, and public libraries (Library catalogue on formal methods). As we can see from the framework in Figure 5-2, buy-in and environments feed into publication. Positive buy-in and a conducive environment should lead to the successful implementation of formal methods, which ought to be publicised. In South Africa, we have an institute called the CSIR (Council for Scientific and Industrial Research), which is a leading technology research organisation in the country. This institute can be used to promote formal methods. The Framework should also be made part of a Formal Methods Body of Knowledge (FMBOK).

RESULTS: After all the steps are followed; positive results should emerge. Even negative results should be made known and lessons learned. Results ought to lead to the development of the systems using formal methods in the practical world. Each successful component should be described and how success was achieved. Results to be documented in a library of re-usable best practices.

5.5 Summary

Chapter 5 presented a framework for the adoption of FMs in the commercial world. The framework highlighted 5 important factors, viz. EDUCATION, BUY-IN, REMUNERATIONS, ENVIRONMENT, TOOL SUPPORT, PUBLICATIONS, and RESULTS. Each factor is also explained in a tabular format above. All of these factors can be executed in parallel, in order to achieve the desired results. A detailed description is also made in a paragraph format. Should the framework be followed and applied practically there ought to be positive changes in the use of FMs. The commercial sector will start to adopt and use formal methods thereby also realising the benefits. This will take time but is possible.

The succeeding chapter will validate the framework, otherwise putting the framework into practice. The validation will be in the form of a case study. From the case study, an explanation is presented on how each step of the conceptual framework could be implemented.

Chapter 6 Framework validation

6.1 Chapter Layout

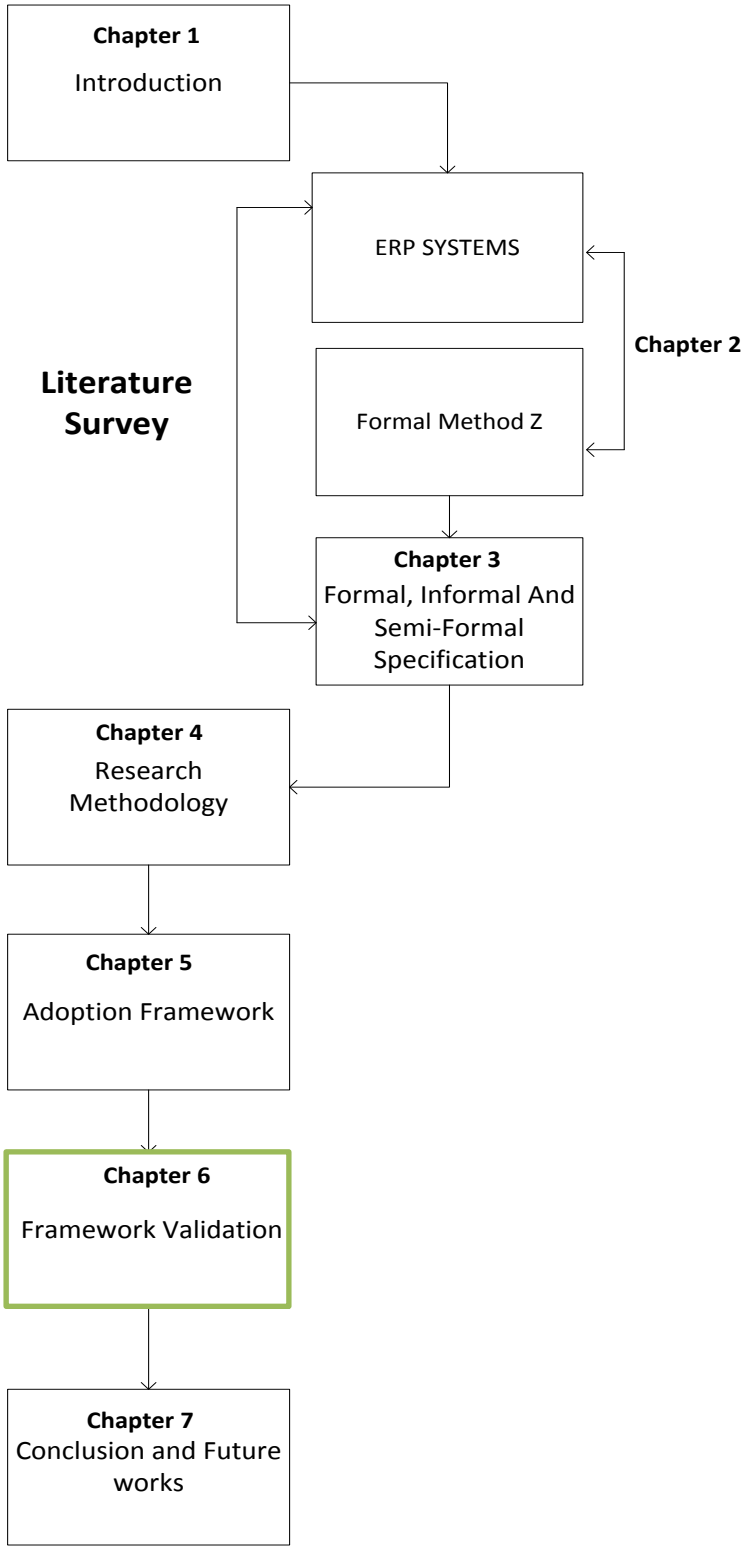


Figure 6-1 Chapter Layout

6.2 Introduction

Previous chapters have explained what formal methods are; the benefits of formal methods; and the research methodologies used to compile this research. Throughout the dissertation, propositions were identified which led to the development of the Formal Methods Adoption Framework. Each step of the framework was explained, also indicating how it is going to increase the adoption of formal methods in the commercial world if followed properly.

Chapter 6 will validate the framework otherwise putting the framework in practice. The validation will be in the form of a case study. From the case study, we explained how each step of the conceptual framework will be implemented.

6.3 Adoption Framework Validation

The diagram below is the formal methods adoption framework, which was presented in the previous chapter.

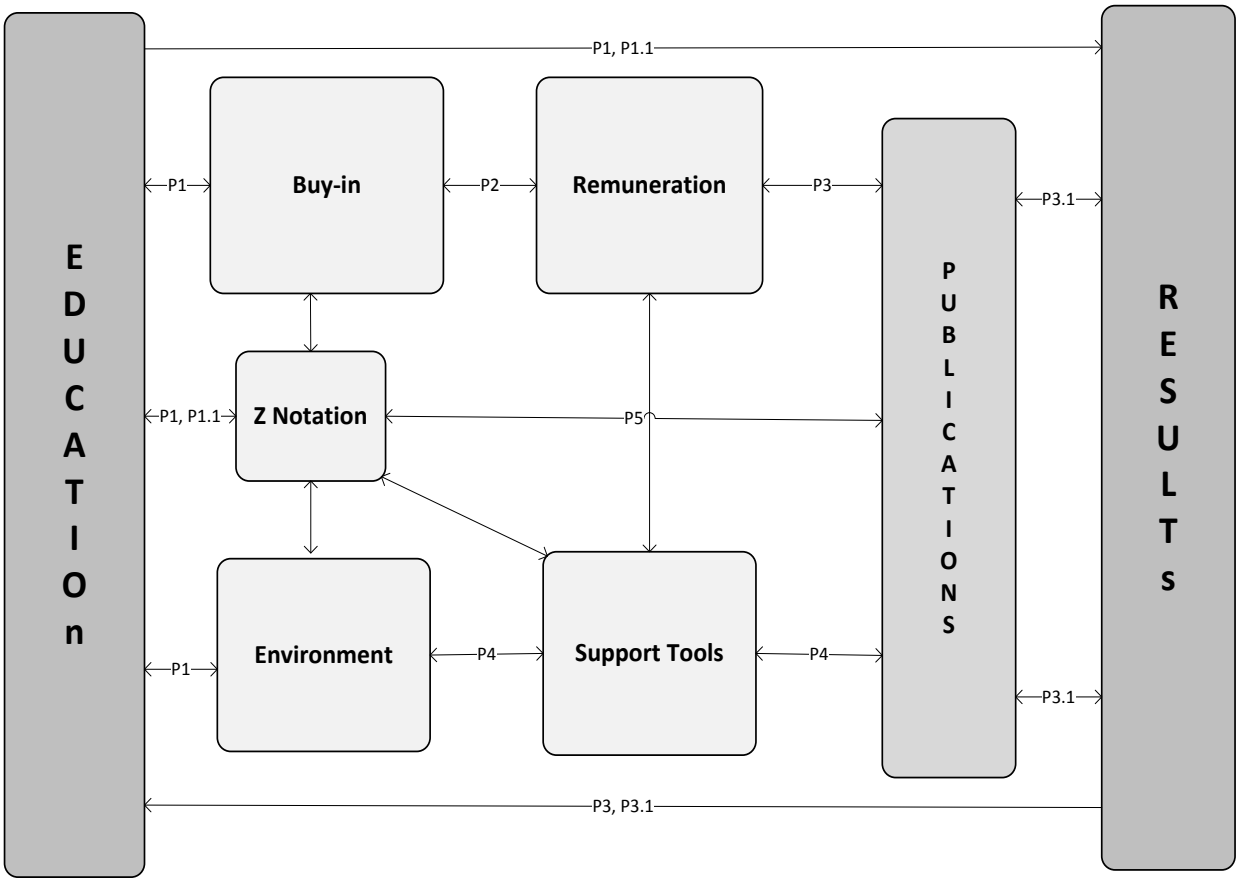


Figure 6-2 Adoption Framework Diagram

Next, I shall conduct a validation of the Figure 6-2 framework on the strength of challenges of a case described next.

6.3.1 Case Study

The South African government is planning to implement an ERP system in some of its key departments. The implementation will be done over a period of years. The government is also in the process of cutting costs and is running under a limited budget. One of the top IT officials suggests that the implementation should include formalisation in order to reduce defects and produce a system that will meet the actual departmental requirements.

There is a shortage of formal methods specialists in South Africa and this has attributed to one of the challenges to the government. Some officials within the government are not buying into the idea of using formal methods to implement an ERP system. In addition to that, not too much information is available in the public domain addressing formal methods; most of the information available is dated and mostly talking about critical systems implementation such as aviation, health systems, and nuclear power plants. Most of the information is from America or the European countries; little to nothing is available from the African countries. The government decided to embark on investing in the education system from the secondary level in order to increase the uptake of formal methods.

Limited off-the-shelf tools to write formal specifications prove to be a challenge coupled with a non-conducive environment, which formal methods are to be used in. Integration between current development methods and formal methods is limited.

The government decided to use the formal methods adoption framework in order to increase the uptake of formal methods and successfully implement the ERP system within various departments.

Notes: *Case Study synthesised by the researcher.*

6.3.2 Formal Methods Adoption Framework Validation

Validation of the formal methods Adoption framework will be linked to the propositions made throughout this dissertation. Each element of the framework will be validated using the above case study, and the importance of each element will be given a percentage, determined qualitatively. The percentage is based on how

much influence the element has towards the adoption of formal methods.

EDUCATION: Education ought to be the pillar of all professions and disciplines. Government develop a policy of implementing formal methods in the current educational system. The policy can be defined as an individual decision, or a collective decision, which will give direction when making future decisions or guide the implementation of the previous decision (Ruano, 2013).

Currently, the following subjects are offered in South Africa schools: languages, mathematics; natural science; life orientation; economic and management sciences; and technology (Republic of South Africa: Department of Higher Education and Training, 2015). As part of technology, a chapter dedicated to formal methods should be introduced. Grade Eight introduces the topic of formal methods and the benefits of using them. Grade Nine to 10 explores greater depths in terms of different languages of formal methods, and how to write them. In Grades 11 to 12, students should be taught how to apply formal methods in practice. After students complete Grade 12, those who choose to pursue a career in computer science and information technology will have a university course dedicated to Formal Methods.

Each university can choose the formal languages they would like to teach or focus upon. At the third-year level, the focus should be on the practicality of formal methods, training students in the commercial world how to apply them. Education is linked to the following proposition 1 and the associated corollary proposition 1.1:

> ***Proposition (Prop) 1****: Education plays a major role in the adoption of formal methods. This includes educating from the high school level to the university level, as well as organisational training in formal methods. Such education plays a pivotal role in the adoption framework.*
>
> ***Prop 1.1****: In addition to the above proposition, formal certificates and diplomas in formal methods ought to be created and awarded to those who qualify. Certification authorities should be well informed about the benefits of formal methods.*

BUY-IN: Buy-in can be described as the acceptance of and commitment to a specific concept or course of action. Some of the students delivered from the education system will become managers or influential people within the commercial world.

Their previous knowledge of formal methods will aid in the adoption of formal methods in the said environment. The concept should filter throughout organisations, and importantly, their IT departments. Buy-in should be from the formal-methods language used and the type of formal techniques adopted.

All government departments implementing the ERP system ought to buy-in to the idea of using formal methods. Management plays a significant role when it comes into influencing change within the department or the organisation. The relevant Government department head will have to go through workshops and high-level training to fully comprehend the benefits of formal methods. Some of the roles that management play pertain to planning, staffing, motivating staff, and implementing change (Partridge and Mintzberg, 2006). Buy-in is linked to the following proposition:

> ***Prop 2:*** *Buy-in from all the business stakeholders is necessary for FMs adoption. Getting top-level management to agree and accept the use of formal methods may well result in the whole organisation adopting formal methods.*

Z NOTATION: Z is the recommended language in this dissertation, owing to its simplicity terms of mathematical set theory and logic and its numerous benefits mentioned throughout this dissertation. Z can be the language of choice to implement the ERP system in government departments. The framework does, however, cater for other formal specification languages. Z learnt from the time of high school education, and throughout university education, will produce a system of high quality with fewer defects as argued in this dissertation. Cost-saving can be achieved due to less re-testing required and defects corrected after implementation. All involved stakeholder in terms of the system requirements will need to have some sort of knowledge around the Z notation. Proposition 6 holds in this regard:

> ***PROP 5:*** *Using the Z notation as an entry language ought to facilitate the adoption of formal methods. Z is believed to be easy to learn and apply. Only basic mathematical set theory and 1st-order logic are required.*

The government will have to run training workshops for all the shareholders interfacing with this project. The government can outsource the training of stakeholders. Training can be at a high-level for any stakeholder who does not

directly implement the project and at a detailed level for all the software engineers and IT staff who are directly responsible for implementing the system. The Z notation is closely linked to support tools and environmental elements. The following proposition holds in this regard:

> ***PROP 4:*** *Tools that are readily available and up to date with the latest technology should facilitate the adoption of FMs. Such tools ought to be integrated with the requirements management software and standard software programming tools e.g., MS Visual Studio.*

ENVIRONMENTS: Environments conducive to the use of formal methods should be created. This can be in the form of the organisation's culture of accepting formal methods. Organisational change can be achieved by receptive formal methods training throughout the affected departments. Integration of the current software development life cycle (SDLC) with formal methods will be required. Formal methods can be applied at any stage of SDLC i.e., from requirement analysis to testing (Pandey and Batra, 2013).

For the government to achieve the above, specialists should be hired for integration and training should be provided throughout for all affected stakeholders. Tools should be available to these environments to produce formal specifications. The environment also involves the psychological aspect of software development. The government should create the right attitude within the team to successfully implement the ERP system. By this, education, buy-in including any FMs technique, would have aided in getting the environments right for the use of formal methods.

Proposition 4 also relates to aspects of the environment in which a system (e.g. an ERP system) is developed.

REMUNERATION: Following the above recommendations, a well-educated formal-methods specialist would be available by now. Some should hold special formal-methods certifications, awarded by accredited bodies. All stakeholders who successfully completed their training ought to be awarded certificates. High remuneration will attract such a specialist to take on this job. South Africa, and as well as Africa more broadly would then have a number of formal-methods specialists. Remuneration can also motivate formal-methods specialists to do better and to encourage others to join. Remuneration also encourages people to stay within the origination for longer periods of time otherwise known as staff retention.

The government should attract well qualified FMs specialists on a permanent basis, or even outsource them. The government should also cater for incentives for example bonuses, extra leave days, pay increases etc. to managers that continuously promote formal methods.

At a more indirect level, proposition 2 may facilitate aspects around remuneration:

> ***PROP 2:*** *Buy-in from all the business stakeholders is necessary for FMs adoption. Getting top-level management to agree and accept the use of formal methods may well result in the whole organisation adopting formal methods.*

SUPPORT TOOLS: Acquiring of tools to aid in using formal methods should be high on the agenda. The government should get the tools that are user-friendly and will (relatively) easily integrate into existing development frameworks such as the .NET framework and the Linux development framework. Tools should allow for automation and facilitate testing and code execution. Part of the support tools will have been introduced as part of the education system. Training provided for the Z notation will also include how to use the support tools. ERP implementation ought to have eased out by now, where there is an increase in the adoption of formal methods in the business world.

Proposition 4 directly support aspects around the use of tools:

> ***PROP 4:*** *Tools that are readily available and up to date with the latest technology should facilitate the adoption of FMs. Such tools ought to be integrated with the requirements management software and standard software programming tools such as MS Visual Studio.*

PUBLICATIONS: The use of publications can be viewed as a process of broadcasting information. The accomplishment of the implementation of the ERP system will be published on the web and public institutions including public libraries. Information will be made accessible to people interested in formal methods, as well as those who would like to become skilled in the use of formal methods. The information will include the lessons learned from the project, the failures and the successes. Implementation ought to be done within a reasonable timeline and within budget. University catalogues will include aspects and publications on formal methods.

To protect the vulnerability of the system and cyber-attacks, sensitive information will be available to those who qualify. The ERP system holds too much information and data where, if all information is made public, the government might lose money and credibility, owing to hackers gaining access to the system and committing a crime. Proposition 5 holds with respect to publications.

Proposition 3.1 supports the publication and related ideas:

- ***PROP 3.1:*** *Widely accepted principles and guidelines on FMs can improve the adoption thereof. Practical, real-world examples of FMs successes and failures must be published in the software engineering and management communities. Publications of formal methods successes in terms of cost savings in projects, clear specifications produced, and the overall final product delivered with fewer defects will raise much interest needed for the adoption of FMs. Practical, real-world examples of FMs successes and failures must be published in the software engineering and management communities.*

RESULTS: Documentation on the results of the entire project, both positive and negative should be produced. Results should be continuously monitored, even while the system is operational. The results element is shown as the last part of the framework, but the results should be noted for all stages/steps of the framework i.e., from education through to publications.

These results will be published in peer-reviewed conference proceedings, journals, libraries, and all other public literature catalogues. Such results will be analysed in order to improve the framework. System performance when it's live/in production and put under stress will be made part of the results.

As with documentation, results pertain to ***PROP 3.1***.

By following all the steps of the framework, either sequentially or in parallel, the government will have implemented the ERP system successfully, on time, and within budget. This will also encourage other stakeholders in the private sector or public sector to adopt formal methods.

6.4 Summary

This chapter validated formal method adoption framework using a case study. The framework was placed in practice and each step/element explained in terms of how it would be implemented.

To conclude the chapter, each step is important in order to achieve successful implementation of systems using formal methods. Education takes greater precedence over other aspects, as it is the foundation of any successful project.

The next chapter will present conclusions, as well as the research findings. Research questions listed in the introduction chapter will be revisited, and an explanation as to how they have been covered and answered throughout the dissertation will be provided. Possible future work in the formal methods field will be suggested.

Chapter 7 Conclusion and future work

7.1 Chapter Layout

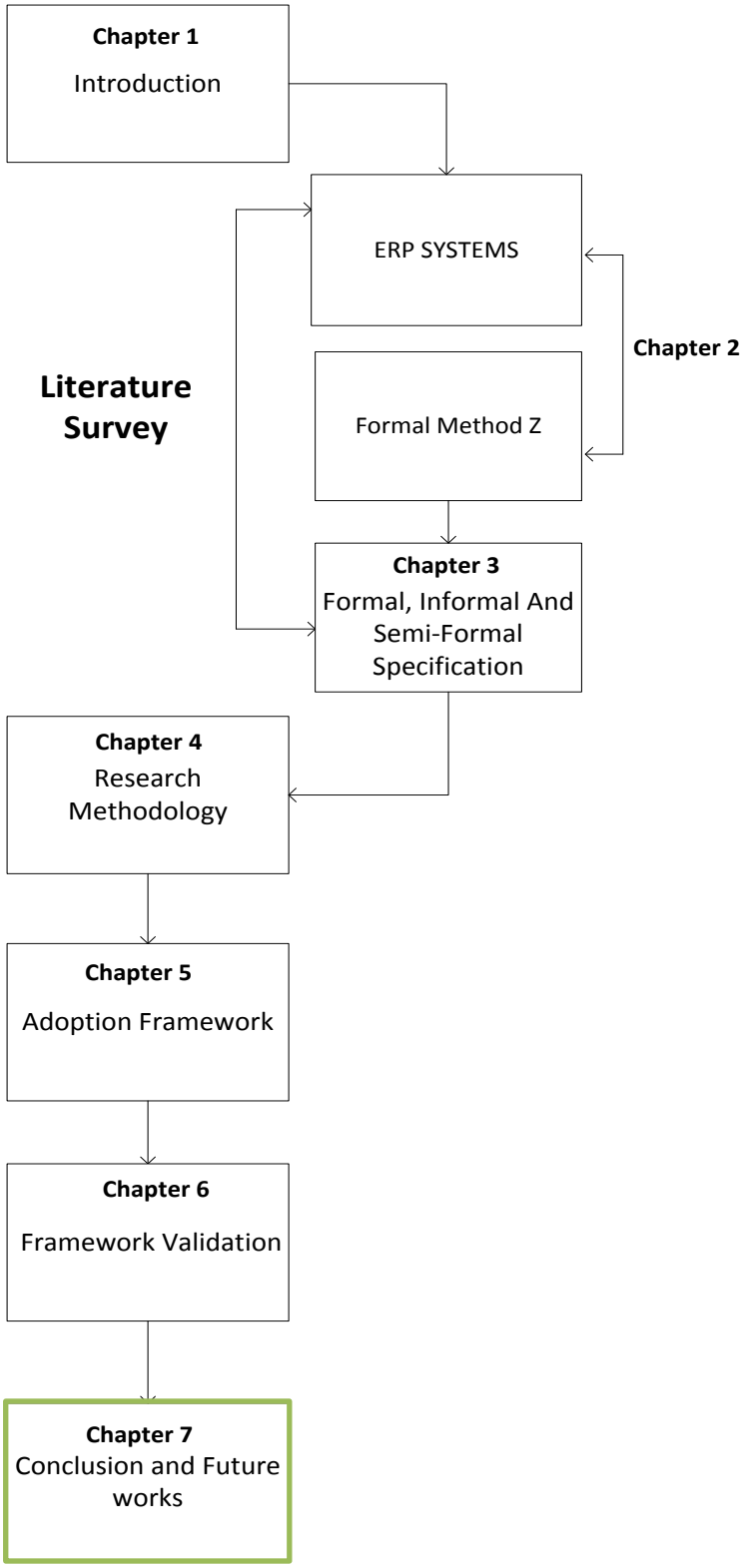


Figure 7-1 Chapter Layout

7.2 Introduction

The previous chapter validated the formal-methods adoption framework using a case study. The framework was placed in a practical case, and each element explained in terms of how it will be implemented. It is anticipated that the framework would facilitate the successful implementation of an ERP system within the government departments.

Chapter 7 revisits the research questions in Chapter One and clarifies how they were answered throughout this dissertation. The chapter also presents the conclusions from the work reported on in this dissertation, as well as the possible future work that may be required in this field.

7.3 Research Questions and Findings

This dissertation has been assessed based on the key formal methods issue, namely the slow adoption of formal methods in the business world. To that extent, the dissertation aimed to develop a mechanism (Framework) that would increase the rate of formal methods adoption if followed. The type of formal language in this research is Z, discussed in earlier chapters. Throughout the dissertation, qualitative propositions were stated which led to the formation of the Formal Methods Adoption Framework. Below is the first question that was raised at the beginning of the research:

RQ 1: *What makes formal methods projects successful?*

This question aims to discover characteristics that make projects that are completed with the use of formal methods successful. This with the hope that successful formal-methods projects will help to inspire businesses that are considering formal methods to adopt and actually use them. Chapter Two has explained what formal methods are, giving an example of companies that have successfully implemented formal methods in their projects. Amazon, the giant online retail store, started using formal methods in 2011, focusing mainly on formal specification and model checking. They have achieved numerous successes and have managed to reduce the number of defects in their critical systems. INTEL also adopted formal methods when they experience problems with their computer chips overheating. This lead to the company saving over 100 million in costs, which they would have otherwise incurred did they not use formal methods. More successful use of formal methods in

business is detailed in Chapter 2.

Arguably, the most important characteristic that makes a formal-methods project successful is that it allows the engineers to rigorously analyse the requirements and write properties about the system. This reduces errors and oversight of the natural language or semi-formal requirements. Formal methods can be costly upfront but the return on investment (ROI) is realised in the long run as less money is spent on correcting errors in the requirements and specification phases. The costs of using formal methods vs. informal methods are also backed up by Figure 3-3 presented in Chapter 3. Design documents can also be produced from the formal specification, as well as test cases. For example, when there is a change in requirements, this will have a minimal impact if formal specifications are used.

The success of every project relies on the people working on it. Having people that have the knowledge and right attitude makes formal methods projects more successful. Buy-in from the team is also important, where people may differ when it comes to the methods chosen, but where compromises are often necessary, so long as there is agreement on a final decision this can result in the success of a team. This is also supported by the formal methods adoption framework, where education, buy-in, support, and other elements are equally important in the successful implementation of software, using formal methods.

> **RQ 1.1** *To what extent can FMs improve on the quality of ERP development?*
>
> The chosen system for this research is the ERP system, due to its criticality within the business. Each organisation has some sort of an ERP system, where they can either be utilising all the modules or some of the modules. Chapter 2 explained what ERP systems are, and the failures to implement them. Chapter 3 then presented a mix of the natural language specification and the semi-formal specification (UML state diagram) of the ERP system. Thereafter, Chapter 3 formalised the ERP system using Z. As an illustration, operations not explicitly shown in the UML were specified in Z.
>
> All the reasons mentioned in the research question one will significantly improve the quality of ERP development and implementation as all the requirements will be clearly understood. This is further demonstrated in Chapter 6 in the form of a case study, where the government is

implementing ERP systems within its departments.

RQ 2: *Why is there a slow adoption of formal method in the commercial world/business?*

Chapter 2 presented the reasons for the slow adoption of formal methods in the commercial world. Because formal methods rely extensively on mathematical notation, this creates the perception that they are difficult to learn and difficult to use. The aforementioned is demonstrated in Chapter 3, where a Z specification is produced for the purchasing module of the ERP system. Schemas for products, orders, customers, and the state of the schemas are presented using set theory and logic. As we saw from the Z specification, for one to be able to read or write it (specify), one must have some knowledge in mathematics and formal logic (Steyn and Van der Poll, 2007). Numerous mathematical notations are used, for example, $\mathbb{P}$, $\oplus$, $\cup$, $\cap$, etc.

Businesses also view the use of formal methods as expensive, as they require an initial investment in the beginning by providing training to the engineers, where the tools to support formal methods are not readily available, making it hard to adopt formal methods in business. Most software vendors do not want to invest in formal methods tools since the market for them is too small, and due to little industrial use of these tools (compared to traditional software development tools), the demand for formal methods remains low (Garavel and Graf, 2013). Currently, formal methods are not integrated into the whole design flow, neither the tools that are currently used.

Most engineers view formal methods as a mechanism that in practice is hard to understand and utilise (Spichkova, 2012b). Most traditional software development techniques are reasonably well-established, and proper standards have been set. Tools supporting those techniques are widely accepted and used in business. Chapter 5 presented the adoption framework, which also recommends that support tools ought to be easily accessible, and the environments to utilise those tools.

To the best of the knowledge of the researcher, no formal accreditation bodies for formal methods, such as IIBA for business analytics and formal methods are active, as they are largely limited to academic projects. With practical examples elaborated

ion n Chapter 2 and Chapter 6, businesses still have the perception that there are not enough practical examples to convince them to adopt Formal Methods. Currently, there is no catalogue of formal methods courses, training, books, and other educational resources (Bourque and Fairley, 2014). Common terminology across all formal methods and language classification is still a challenge, similar to which formal technique to use when. Some formal methods researchers are hesitant to take part in the development of real-world systems. The reasons might be they are wary of failures and the pressure that comes with developing systems in the commercial world. In Chapter 5, the adoption framework recommends that the results of successful projects be made public.

RQ 2.1: *What is the status quo of the use of FMs in the commercial world/business?*

Currently, there is little to no use of formal methods in business. Many engineers working for large corporations within the IT division have little knowledge to, no knowledge at all of FMs. The aforementioned is still a trend, despite all the benefits and successes presented in this dissertation, and many other research papers and publications.

The famous advocate of formal methods, Hall (2007, p. 5) stated “I am an enthusiast for formal methods, and I can show that they offer clear benefits. However, these benefits are not automatic — they depend on intelligent application of methods where they can add value.” Naturally, the use of intelligent FMs tools will go a great distance in support of Hall’s advocacy.

The business world is profit driven, and they do not have the appetite to invest in formal methods. As indicated, formal methods are still viewed as difficult to use and companies see little reason to invest in them. Chapter 2 also helps to answer this question by mentioning various examples or scenarios in business. It seems as if this status quo towards formal methods is not going to change anytime soon. Yet, a comprehensive industry survey would shed more light on the use of FMs in the South African software industry. This can be part of future work (refer below).

RQ3: *What can be done to increase the adoption of formal methods in the commercial world/Business?*

The use of formal methods remains mainly on research, where the research community needs to make formal methods more practical and easier to use. Knowledge is needed to facilitate an increase in the use of formal methods. Baier and Katoen (2010) state that “FMs should be part of the education of every computer scientist and software engineer, just as the appropriate branch of applied maths is a necessary part of the education of all other engineers.” Formal methods should also be introduced as part of the syllabus as early as high school to students interested in learning software engineering. Proper FMs certification needs to be established and awarded to individuals who qualify to use formal methods. This is demonstrated in Chapter 5, where education is vital.

Adequate remuneration for the formal methods specialist will hopefully increase the number of engineers who might be interested in studying them. Also, formal accreditation in formal methods needs to be established. The benefits of formal methods and any successful formal method project ought to be presented and made public. This can be achieved via the use of weekly newsletters, or research websites such as Gartner, publishing stories of companies that have successfully developed and implemented systems using formal methods.

Proper tools for formal methods should be produced, i.e. tools checking the syntax and automatic provers. Ideally, a tool able to automatically produce code from the formal specification will increase the confidence from business as well as engineers. Chapter 5 presented a formal-methods framework to be adopted and widely accepted by the software engineering community and business alike. A research paper about the formal-methods adoption framework was written and published in an IRED conference (Nemathaga and Van der Poll, 2019). As indicated, it is hoped that the said framework will over time increase the adoption of formal methods in business. Formal methods should also be classified based on their tools and language, where relationships among formal methods need to be established and readily available.

7.4 Research Summary

This research examined the reasons why there is slow adoption of formal methods in the business/commercial world. The research further investigated what can possibly be done to fasten the adoption of formal methods within the business or the commercial world. This led to the development of the Formal Methods Adoption Framework. The Framework was validated and put into practice using the case

study. The choice of the system for this research is the ERP system, due to its criticality within the business or commercial world. This research also made reference to the businesses/companies that have successfully use formal methods as part of their software development process.

With modern day technology, more and more businesses are becoming reliant on IT solutions to automate their business processes. The development of software that is free from errors, or at least highly dependable is desirable (Fisher, 2011). Sommerville (2005) also states that modern systems have complex requirements, ensuring the successful completion of the project within timeline and on budget, where correct software engineering practices ought to be adhered to at all times. When compared to conventional design methods formal methods allow for the development of high-end systems, using (discrete) mathematics, resulting in a final system with a reduced number of errors. The current process of quality assurance or testing only reveals current errors on the system but does not show that there are no errors (absence of errors) in the system (Boca, Siddiqi and Bowen, 2010).

Formal methods have shown themselves to be beneficial, but still, there is a slow adoption in the commercial world. Formal methods are still viewed as difficult to use owing to the underlying mathematical formalism. There are many myths that surround the use of formal methods, such that they guarantee a perfect system. More research needs to be undertaken (refer Section 7.5 below) and will require the involvement of both the public and private sector to promote the use of formal methods. Formal methods education should be introduced at the early stages of the education system, where the benefits of formal methods ought to be made public for the commercial world so as to be able to access this information.

7.5 Future Work

This dissertation does not solve all the problems around formal methods, which methods have been around for years, and this dissertation focuses on what can be done to increase the usage of formal methods in the commercial world. The research in this dissertation is interpretive in nature and is mostly based on scholarly works and industrial reports that have already been done on this topic i.e., books, e-journals, case studies, and work interviews.

Future work in this area could be pursued along a number of avenues.

Common terminology needs to be developed across the formal methods field. This needs to be widely accepted and standardised. The issue of formal-methods tools has been cited by most researchers, where tools need to be developed, especially user-friendly automatic provers to establish properties and consequences of a formal specification. Tools that are already developed are either not user-friendly, or do not perform all the required functionality to produce correct formal specifications. These tools need to be integrated into current development frameworks, such as .NET and JAVA, to name a few. Automatic conversion of first-order logic statements to a full Z specification needs to be investigated. Tools need to be classified and demos of the tools in a form of videos, also indicating the strength of it in practice must be made available.

The formal-methods adoption framework was developed and validated, both via qualitative means. It is vital that the framework should be further validated by exercising it in one or more companies in the industry to determine its scalability, and it should also be validated through quantitative (statistical) means, aimed at deriving a model from the framework. More practical examples are needed, specifying the advantages and disadvantages of different design methodologies.

Other aspects to be further researched include:

- Validation of a formal specification by executing the specification or simulating it to show its behaviour. Some work on this has been done, e.g. running a Z specification in Prolog, yet it's a tedious process.
- Integration of formal notations with more widely used notations such as use-case diagrams, UML class diagrams, collaboration diagrams, etc.
- Automation of formal descriptions so as to generate test cases and even code/scripts. The automatic transformation of a formal specification into a high-level language, coupled with proof obligations to be discharged at each transformation iteration needs further research.

A comprehensive industry survey ought to be done on the use of FMs in the SA software industry.

References

Adesina-Ojo, A. A., Van Der Poll, J. A. and Venter, L. M. (2011). Towards the formalisation of object-oriented methodologies. SAICSIT 2011: *ACM International Conference Proceeding Series*, University of South Africa, Pretoria. [Online]. Available at: doi:10.1145/2072221.2072252.

Alagar, V. S. and Periyasamy, K. (2011). *Specification of Software Systems*, Texts in Computer Science. London : Springer London. [Online]. Available at: doi:10.1007/978-0-85729-277-3.

Alsmadi, I. (2017). *Using formal method for gui model verification*. (January 2008), Yarmouk University.

Andrew, S. and Halcomb, E. J. (2009). *Mixed Methods Research for Nursing and the Health Sciences*. Andrew, S. and Halcomb, E. J. (Eds). Oxford, UK : Wiley-Blackwell. [Online]. Available at: doi:10.1002/9781444316490.

Asgar, T. S. and King, T. M. (2016). Formalizing Requirements in ERP Software Implementations. *Lecture Notes on Software Engineering*, 4 (1), pp.34–40. [Online]. Available at: doi:10.7763/lnse.2016.v4.220.

Atlee, J. M., Beidu, S., Day, N. A., Faghih, F. and Shaker, P. (2013). Recommendations for improving the usability of formal methods for product lines. In: *2013 1st FME Workshop on Formal Methods in Software Engineering, FormaliSE 2013 - Proceedings*. 2013. pp.43–49. [Online]. Available at: doi:10.1109/FormaliSE.2013.6612276.

Babu, K. V. S. . and Bezawada, M. (2012). Enterprise Resource Planning. *SSRN*. [Online]. Available at: doi:10.2139/ssrn.2142346.

Banerjee, P., Sarkar, A. and Debnath, N. C. (2016). Modeling component interaction: Z - Notation based approach. In: *2015 International Conference on Computing, Management and Telecommunications, ComManTel 2015*. 2016. [Online]. Available at: doi:10.1109/ComManTel.2015.7394261.

Beck, K., Beedle, M., Bennekum, A. Van, Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., et al. (2001). *Manifesto for Agile Software Development*. [Online]. Available at: http://agilemanifesto.org/.

Benjamin, M. (1990). *A Message Passing System. An example of combining CSP and Z*. In: pp.221–228. [Online]. Available at: doi:10.1007/978-1-4471-3877-8_15.

Bernroider, E. W. N., Wong, C. W. Y. and Lai, K. hung. (2014). From dynamic capabilities to ERP enabled business improvements: The mediating effect of the implementation project. *International Journal of Project Management*. [Online]. Available at: doi:10.1016/j.ijproman.2013.05.006.

Bjørner, D. and Havelund, K. (2014). *40 Years of Formal Methods*. In: Springer, Cham. pp.42–61. [Online]. Available at: doi:10.1007/978-3-319-06410-9_4.

Boca, P., Siddiqi, J. I. and Bowen, J. P. (2010). *Formal Methods: State of the Art and New Directions*. 1st ed. Boca, P., Bowen, J. P. and Siddiqi, J. (Eds). London : Springer London. [Online]. Available at: doi:10.1007/978-1-84882-736-3.

Boehm, B. and Turne, R. (2015). The incremental commitment spiral model (ICSM): principles and practices for successful systems and software. *In Proceedings of the 2015 International Conference on Software and System Process (ICSSP 2015). ACM, New York, NY, USA, 175-176.* [Online]. Available at: doi:DOI=http://dx.doi.org/10.1145/2785592.2785619.

Börger Egon and Stärk Robert. (2003). Abstract State Machines. *Springer-Verlag Berlin Heidelberg*, p.437. [Online]. Available at: doi:10.1007/978-3-642-18216-7.

Bottoni, P. and Fish, A. (2011). Policy specifications with Timed Spider Diagrams. In: *Proceedings - 2011 IEEE Symposium on Visual Languages and Human Centric Computing, VL/HCC 2011*. 2011. pp.95–98. [Online]. Available at: doi:10.1109/VLHCC.2011.6070385.

Bourque, P. and Fairley, R. E. (2014). *SWEBOK v.3 - Guide to the Software Engineering - Body of Knowledge.* [Online]. Available at: doi:10.1234/12345678.

Bowen, J. P. (1996). Formal Specification and Documentation using Z: A Case Study Approach. *Citeseer*. [Online]. Available at: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.14.8627&rep=rep1&type=pdf.

Bowen, J. P. (2016). The Z Notation: Whence the Cause and Whither the Course? In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. 9506. Springer, Cham. pp.103–151. [Online]. Available at: doi:10.1007/978-3-319-29628-9_3.

Bowen, J. P. and Hinchey, M. G. (1995). Ten commandments of formal methods. *Computer*, 28 (4), pp.56–63. [Online]. Available at: doi:10.1109/2.375178.

Bowen, J. P. and Hinchey, M. (2012). Ten commandments of formal methods… Ten years on. In: *Conquering Complexity*. pp.237–251. [Online]. Available at: doi:10.1007/978-1-4471-2297-

5_11.

Cansell, D. and Méry, D. (2003). Foundations of the B method. *Computing and Informatics*, 22 (3–4), pp.221–256. [Online]. Available at: https://cai.type.sk/content/2003/3-4/foundations-of-the-b-method/.

Cardoso, J., Bostrom, R. P. and Sheth, A. (2004). Workflow Management Systems and ERP Systems: Differences, Commonalities, and Applications. *Information Technology and Management*. [Online]. Available at: doi:10.1023/b:item.0000031584.14039.99.

Carlier, M., Dubois, C. and Gotlieb, A. (2012). *FM 2012: Formal Methods*, Lecture Notes in Computer Science. Giannakopoulou, D. and Méry, D. (Eds). Berlin, Heidelberg : Springer Berlin Heidelberg. [Online]. Available at: doi:10.1007/978-3-642-32759-9.

Charette, R. N. (2005). Why Software Fails. *IEEE Spectrum*, 42 (9), pp.42–49. [Online]. Available at: doi:10.1109/MSPEC.2005.1502528.

Chow, S. and Ruskey, F. (2004). Drawing Area-Proportional Venn and Euler Diagrams. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. 2912. Springer, Berlin, Heidelberg. pp.466–477. [Online]. Available at: doi:10.1007/978-3-540-24595-7_44.

Coates, C. (2012). UML 2 Class Diagram Tutorial. *Sparx Systems*.

Cousineau, D., Doligez, D., Lamport, L., Merz, S., Ricketts, D. and Vanzetto, H. (2012). TLA + proofs. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. 7436 LNCS. 2012. pp.147–154. [Online]. Available at: doi:10.1007/978-3-642-32759-9_14.

Crepaldi, G. (2005). *Ninth Edition of the G.B. Morgagni Awards Program*. [Online]. Available at: doi:10.1111/j.1365-2362.2005.01463.x.

Davis, J. A., Clark, M., Cofer, D., Fifarek, A., Hinchman, J., Hoffman, J., Hulbert, B., Miller, S. P. and Wagner, L. (2013). Study on the barriers to the industrial adoption of formal methods. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. 2013. [Online]. Available at: doi:10.1007/978-3-642-41010-9_5.

Dehbonei, B. and Mejia, F. (2012). *Formal methods in the railways signalling industry*. [Online]. Available at: doi:10.1007/3-540-58555-9_84.

Di Vito, B. (2014). *Digital Avionics Handbook, Third Edition*. Spitzer, C., Ferrell, U. and Ferrell,

T. (Eds). CRC Press. [Online]. Available at: doi:10.1201/b17545.

Dongmo, C. (2016). *FORMALISING NON-FUNCTIONAL REQUIREMENTS EMBEDDED IN USER REQUIREMENTS NOTATION (URN) MODELS*. PhD Thesis, University of South Africa. [Online]. Available at: http://hdl.handle.net/10500/23395.

Dongmo, C. and van der Poll, J. A. (2010). A four-way framework for validating a specification. In: *Proceedings of the 2010 Annual Research Conference of the South African Institute of Computer Scientists and Information Technologists on - SAICSIT '10*. 2010. New York, New York, USA : ACM Press. pp.48–57. [Online]. Available at: doi:10.1145/1899503.1899509.

Dongmo, C. and van der Poll, J. A. (2011). Evaluating software specifications by comparison. In: *Proceedings of the South African Institute of Computer Scientists and Information Technologists Conference on Knowledge, Innovation and Leadership in a Diverse, Multidisciplinary Environment - SAICSIT '11*. 2011. New York, New York, USA : ACM Press. p.87. [Online]. Available at: doi:10.1145/2072221.2072232.

Easterbrook, S., Lutz, R. and Covington, R. (1998). Experiences using lightweight formal methods for requirements modeling. *IEEE Transactions on Software Engineering*. [Online]. Available at: doi:10.1109/32.663994.

Elbertsen, L. and Reekum, R. Van. (2008). To ERP or not to ERP? Factors influencing the adoption decision. *International Journal of Management and Enterprise Development*. [Online]. Available at: doi:10.1504/IJMED.2008.017434.

Enderton, H. B. (1977). Elements of Set Theory. *Information Storage and Retrieval*, 58 (303), p.279. [Online]. Available at: doi:10.2307/2282754.

Equey, C., Kusters, R. J., Varone, S. and Montandon, N. (2008). Empirical study of ERP systems implementation costs in Swiss SMES. *International Conference on Enterprise Information Systems (ICEIS 2008)*, pp.143–148. [Online]. Available at: http://campus.hesge.ch/equeyc/doc/ERP/ERP_impl_costs_swiss.pdf.

Felderer, M., Piazolo, F., Ortner, W., Brehm, L. and Hof, H. J. (2016). Innovations in enterprise information systems management and engineering: 4th international conference, ERP Future 2015 - research munich, Germany, November 16-17, 2015 revised papers. In: *Lecture Notes in Business Information Processing*. 245. 2016. [Online]. Available at: doi:10.1007/978-3-319-32799-0.

Fenton, N. E. and Neil, M. (2000). Software metrics. In: *Proceedings of the conference on The future of Software engineering - ICSE '00*. 2000. New York, New York, USA : ACM Press.

pp.357–370. [Online]. Available at: doi:10.1145/336512.336588.

Fisher, D. W. (1990). The Human Genome—No Less! *Hospital Practice*, 25 (2), pp.13–13. [Online]. Available at: doi:10.1080/21548331.1990.11703905.

Fisher, M. (2011). *An Introduction to Practical Formal Methods Using Temporal Logic*. [Online]. Available at: doi:10.1002/9781119991472.

Fix, L. (2008). Fifteen years of formal property verification in intel. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. 5000 LNCS. 2008. pp.139–144. [Online]. Available at: doi:10.1007/978-3-540-69850-0_8.

Gabbar, H. A. (2006). *Modern formal methods and applications*. [Online]. Available at: doi:10.1007/1-4020-4223-X.

Gao, J., Zhang, L. and Wang, Z. (2008). Decision support in procuring requirements for ERP software. In: *Proceedings of the 9th International Conference for Young Computer Scientists, ICYCS 2008*. 2008. pp.1126–1131. [Online]. Available at: doi:10.1109/ICYCS.2008.158.

Garavel, H. and Graf, S. (2013). Formal Methods for Safe and Secure Computer Systems. *Federal Office for Information Security*. [Online]. Available at: https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/Studien/formal_methods_study_875/formal_methods_study_875.pdf?__blob=publicationFile&v=2.

George, V. and Vaughn, R. (2003). Application of lightweight formal methods in requirement engineering. Crosstalk. *The Journal of Defence Software Engineering.*

Gieryn, T. F. and Giddens, A. (2006). Positivism and Sociology. *Contemporary Sociology*, 5 (5), p.665. [Online]. Available at: doi:10.2307/2063389.

Gilliam, D. P., Powell, J. D. and Bishop, M. (2005). Application of lightweight formal methods to software security. In: *Proceedings of the Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises, WETICE*. 2005. [Online]. Available at: doi:10.1109/WETICE.2005.19.

Glass, R. L. . (1996). Formal Methods are a Surrogate for a More Serious Software Concern. *IEEE Computer*, 29 (4), p.19.

Hall, A. (2007). Realising the benefits of formal methods. In: *Journal of Universal Computer Science*. 13 (5). Springer, Berlin, Heidelberg. pp.669–678. [Online]. Available at: doi:10.1007/11576280_1.

Hamilton-Smith, E. (2001). Book Review: Doing Qualitative Research: A Practical Handbook. *Sociological Research Online*, 5 (4), pp.131–132. [Online]. Available at: doi:10.1177/136078040100500415.

Hammer Michael. (2003). Reengineering Work: Don't Automate, Obliterate. *Harvard BUsiness Review*, (July-August), pp.1–19. [Online]. Available at: https://hbr.org/1990/07/reengineering-work-dont-automate-obliterate.

Harrison, J. (2003). Formal verification at Intel. In: *18th Annual IEEE Symposium of Logic in Computer Science, 2003. Proceedings.* 2003. IEEE Comput. Soc. pp.45–54. [Online]. Available at: doi:10.1109/LICS.2003.1210044.

Harrison, J. (2010). Formal Methods at Intel — An Overview. *Symposium A Quarterly Journal In Modern Foreign Literatures*, 2010 (April), pp.0–10.

Hastie, S. and Wojewoda, S. (2015). Standish Group 2015 Chaos Report - Q&A with Jennifer Lynch. *Http://Www.Infoq.Com/Articles/Standish-Chaos-2015*, pp.1–9. [Online]. Available at: http://www.infoq.com/articles/standish-chaos-2015.

Hussain, S., Dunne, P. and Rasool, G. (2013). Formal Specification of Security Properties using Z Notation. *Research Journal of Applied Sciences, Engineering and Technology*, 5 (19), pp.4664–4670. [Online]. Available at: doi:10.19026/rjaset.5.4298.

Iddiqui, M. U. A. J. S., Akhter, M. S. and Ian, N. A. A. L. I. M. (2014). *a Comparative Analysis of Conventional Software Development Approaches Vs . Formal Methods in Call Distribution Systems*. Vawkum Transaction on Computer Sciences.

IIBA. (2015). *BABOK*. Toronto, Ontario, Canada .

Ilić, D. (2007). Deriving formal specifications from informal requirements. In: *Proceedings - International Computer Software and Applications Conference*. 2007. [Online]. Available at: doi:10.1109/COMPSAC.2007.104.

Jacky, J. (2004). Formal specification for a clinical cyclotron control system. *ACM SIGSOFT Software Engineering Notes*, 15 (4), pp.45–54. [Online]. Available at: doi:10.1145/99571.99814.

Jaspan, C., Keeling, M., Maccherone, L., Zenarosa, G. L. and Shaw, M. (2009). Software Mythbusters Explore Formal Methods. *IEEE Software*, 26 (6), pp.60–63. [Online]. Available at: doi:10.1109/MS.2009.188.

Kaivola, R. (2011). Intel ® Core ™ i7 processor execution engine validation in a functional language based formal framework. In: *Lecture Notes in Computer Science (including subseries*

Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). 6539 LNCS. 2011. p.1. [Online]. Available at: doi:10.1007/978-3-642-18378-2_1.

Karibskii, A. V. (1991). Managing the development of large-scale systems. *Mathematics and Computers in Simulation*, 33 (4), pp.287–293. [Online]. Available at: doi:10.1016/0378-4754(91)90107-E.

Kilic, H. S., Zaim, S. and Delen, D. (2015). Selecting 'the best' ERP system for SMEs using a combination of ANP and PROMETHEE methods. *Expert Systems with Applications*, 42 (5), pp.2343–2352. [Online]. Available at: doi:10.1016/j.eswa.2014.10.034.

Kneuper, R. (1997). Limits of formal methods. *Formal Aspects of Computing*, 9 (4), pp.379–394. [Online]. Available at: doi:10.1007/BF01211297.

Kotonya, G. and Sommerville, I. (1998). Requirements Engineering : Processes and Techniques. *Star*, p.294. [Online]. Available at: http://www.amazon.com/Requirements-Engineering-Processes-Techniques-Worldwide/dp/0471972088.

Krause, J., Hintze, E., Magnus, S. and Diedrich, C. (2012). Model Based Specification, Verification, and Test Generation for a Safety Fieldbus Profile. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. 7612 LNCS. Springer, Berlin, Heidelberg. pp.87–98. [Online]. Available at: doi:10.1007/978-3-642-33678-2_8.

Kwahk, K. Y. and Ahn, H. (2010). Moderating effects of localization differences on ERP use: A socio-technical systems perspective. *Computers in Human Behavior*, 26 (2), pp.186–198. [Online]. Available at: doi:10.1016/j.chb.2009.10.006.

Lalena, J. N., Cushing, B. L., Falster, A. U., Simmons, W. B., Seip, C. T., Carpenter, E. E., O'Connor, C. J. and Wiley, J. B. (1998). A Multistep Topotactic Route to the New Mixed-Valence Titanate, Na(2)(-)(x)()(+)(y)()Ca(x)()(/2)La(2)Ti(3)O(10). Electron Localization Effects in a Triple-Layered Perovskite. *Inorganic chemistry*, 37 (18), pp.4484–4485. [Online]. Available at: doi:10.1021/ic980611g.

Lorch, J. R., Chen, Y., Kapritsos, M., Parno, B., Qadeer, S., Sharma, U., Wilcox, J. R. and Zhao, X. (2020). Armada: low-effort verification of high-performance concurrent programs. In: Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation. 11 June 2020. New York, NY, USA : ACM. pp.197–210. [Online]. Available at: doi:10.1145/3385412.3385971.

Laroussinie, F. (2010). CHRISTEL BAIER AND JOOST-PIETER KATOEN * Principles of Model

Checking. MIT Press (May 2008). * ISBN: 978-0-262-02649-9. 44.95. 975 pp. Hardcover. *The Computer Journal*, 53 (5), pp.615–616. [Online]. Available at: doi:10.1093/comjnl/bxp025.

Li, F.-L., Horkoff, J., Borgida, A., Guizzardi, G., Liu, L. and Mylopoulos, J. (2015). From Stakeholder Requirements to Formal Specifications Through Refinement. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. 9013. pp.164–180. [Online]. Available at: doi:10.1007/978-3-319-16101-3_11.

Lions, J. L. (1996). Ariane 5 Flight 501 Failure. *Report by the Inquiry Board Paris*, 19 (July), p.1. [Online]. Available at: http://esamultimedia.esa.int/docs/esa-x-1819eng.pdf.

Liu, S. and Adams, R. (1995). Limitations of Formal Methods and An Approach to Improvement. *Faculty of Information Sciences Hiroshima City University*, pp.498–507. [Online]. Available at: doi:10.1109/APSEC.1995.496999.

Liu, S., Stavridou, V. and Dutertre, B. (1995). The practice of formal methods in safety-critical systems. *The Journal of Systems and Software*, 28 (1), pp.77–87. [Online]. Available at: doi:10.1016/0164-1212(94)00082-X.

Lockhart, J., Purdy, C. and Wilsey, P. (2014). Formal methods for safety critical system specification. In: *2014 IEEE 57th International Midwest Symposium on Circuits and Systems (MWSCAS)*. August 2014. IEEE. pp.201–204. [Online]. Available at: doi:10.1109/MWSCAS.2014.6908387.

Lorch, J. R., Chen, Y., Kapritsos, M., Parno, B., Qadeer, S., Sharma, U., Wilcox, J. R. and Zhao, X. (2020). *Armada: low-effort verification of high-performance concurrent programs*. In: 2020. [Online]. Available at: doi:10.1145/3385412.3385971.

Ma, Z. M. (2008). Fuzzy Conceptual Information Modeling in UML Data Model. In: *2008 International Symposium on Computer Science and Computational Technology*. 2. 2008. IEEE. pp.331–334. [Online]. Available at: doi:10.1109/ISCSCT.2008.353.

Malik, P. and Utting, M. (2005). CZT: A Framework for Z Tools. In: *Lecture Notes in Computer Science*. 3455. Springer, Berlin, Heidelberg. pp.65–84. [Online]. Available at: doi:10.1007/11415787_5.

Markus, M. L., Tanis, C. and Van Fenema, P. C. (2000). Multisite ERP implementations. *Communications of the ACM*, 43 (4), pp.42–46. [Online]. Available at: doi:10.1145/332051.332068.

Moremedi, K. and van der Poll, J. A. (2013). Transforming Formal Specification Constructs into Diagrammatic Notations. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. 8216 LNCS. Springer, Berlin, Heidelberg. pp.212–224. [Online]. Available at: doi:10.1007/978-3-642-41366-7_18.

Moremedi, K. and van der Poll, J.A. (2019). Towards a Comparative Evaluation of Text-Based Specification Formalisms and Diagrammatic Notations, *International Journal of Data Mining, Modelling and Management* (*IJDMMM)*, 11(3), pp. 259 – 283, Inderscience Enterprises Ltd.

Nathan, A. J. and Scobell, A. (2012). *How China sees America*. [Online]. Available at: doi:10.1017/CBO9781107415324.004.

Nemathaga, A. and van der Poll, J. A. (2019). Adoption of Formal Methods in the Commercial World. In: *Eighth International Conference on Advances in Computing, Communication and Information Technology CCIT*. 2019. Birmingham City University, Birmingham, United Kingdom : Institute of Research Engineers and Doctors. pp.75–84. [Online]. Available at: doi:10.15224/978-1-63248-169-6-12.

Nerode, A. and Shore, R. A. (1997). Introduction. In: *Logic for Applications*. New York, NY : Springer New York. pp.1–5. [Online]. Available at: doi:10.1007/978-1-4612-0649-1_1.

Newcombe, C., Rath, T., Zhang, F., Munteanu, B. and al, et. (2013). Use of Formal Methods at Amazon Web Services. *Research.Microsoft.Com*. [Online]. Available at: http://research.microsoft.com/en-us/um/people/lamport/tla/formal-methods-amazon.pdf%5Cnpapers2://publication/uuid/7887148C-66F4-4995-A00E-401C20C7F57D.

Newcombe, C., Rath, T., Zhang, F., Munteanu, B., Brooker, M. and Deardeuff, M. (2015). How Amazon web services uses formal methods. *Communications of the ACM*, 58 (4), pp.66–73. [Online]. Available at: doi:10.1145/2699417.

Norris, C. (2005). Epistemology. *London: Continuum*.

Oates, B. J. (2006). *Research Information Systems and Computing.* London : SAGE Publications Ltd.

Ogheneovo, E. E. (2014). Software Dysfunction: Why Do Software Fail? *Journal of Computer and Communications*, 02 (06), pp.25–35. [Online]. Available at: doi:10.4236/jcc.2014.26004.

Pagliari, C. (2007). Design and evaluation in eHealth: challenges and implications for an interdisciplinary field. *Journal of medical Internet research*, 9 (2), IEEE., p.e15. [Online]. Available at: doi:10.2196/jmir.9.2.e15.

Palmquist, M. S., Lapham, M. A., Miller, S., Chick, T. and Ozkaya, I. (2013). *Parallel Worlds: Agile and Waterfall Differences and Similarities*. Software Engineering Institute. [Online]. Available at: doi:CMU/SEI-2013-TN-021.

Pandey, K.S. (2016a). Research Methodology. In: *JISTEM Journal of Information Systems and Technology Management*. 6 (2). pp.111–127. [Online]. Available at: doi:10.1007/978-81-322-2785-4_4.

Pandey, K.S. and Batra, M. (2013). Formal Methods in Requirements Phase of SDLC. *International Journal of Computer Applications*, 70 (13), pp.7–14. [Online]. Available at: doi:10.5120/12020-8017.

Pandey, K. N. (2016b). Research Methodology. In: *Studies in Systems, Decision and Control*. Springer, New Delhi. pp.111–127. [Online]. Available at: doi:10.1007/978-81-322-2785-4_4.

Pang, A. (2016). *Top 10 ERP Software Vendors and Market Forecast 2015-2020*. [Online]. Available at: https://www.appsruntheworld.com/top-10-erp-software-vendors-and-market-forecast-2015-2020/.

Parnas, D. L. (2010). Really rethinking 'formal methods'. *Computer*, 43 (1), pp.28–34. [Online]. Available at: doi:10.1109/MC.2010.22.

Partridge, B. E. and Mintzberg, H. (2006). The Nature of Managerial Work. *Operational Research Quarterly (1970-1977)*. [Online]. Available at: doi:10.2307/3007945.

Pressman, R. S. (2009). *Software Engineering A Practitioner's Approach 7th Ed - Roger S. Pressman*. [Online]. Available at: doi:10.1017/CBO9781107415324.004.

Republic of South Africa: Department of Higher Education and Training. (2015). Summary for Policymakers. In: Intergovernmental Panel on Climate Change (Ed). *Climate Change 2013 - The Physical Science Basis*. Cambridge : Cambridge University Press. pp.1–30. [Online]. Available at: doi:10.1017/CBO9781107415324.004.

Royce, D. W. W. (1970). *Managing the Development of large Software Systems*. [Online]. Available at: doi:1016/0378-4754(91)90107-E.

Ruano, L. (2013). *The Europeanization of national foreign policies towards Latin America*. Ruano, L. (Ed). Routledge. [Online]. Available at: doi:10.4324/9780203100899.

Saunders and Lewis. (2014). *Choosing a research design (Part 1): Philosophies and approaches*. [Online]. Available at: http://futureideas.eu/wp-content/uploads/2014/01/Figure-5-IT-SERVICES-OFFFFSHORING-IN-MOROCCO.jpg.

Saunders, M.N. K., Lewis, P., Thornhill, A. and Bristow, A. (2015). Understanding research philosophy and approaches to theory development. In: Saunders, Mark N. K.; Lewis, Philip and Thornhill, Adrian eds. *Research Methods for Business Students.* Harlow: Pearson Education, pp. 122–161.

Schach, S. . (2011). *Object-Oriented & Classical Software Engineering*. [Online]. Available at: doi:10.1036/0072554509.

Schneider, K. (2004). *Verification of Reactive Systems: Formal Methods and Algorithms*. Springer Science & Business Media. [Online]. Available at: http://books.google.com/books?hl=en&lr=&id=Z92bL1VrD_sC&pgis=1.

Scott, T. (2000). ML tutorial. *Journal of Computing Sciences in Colleges*, 16 (1), p.249.

Sengupta, S. and Bhattacharya, S. (2006). Formalization of UML use case diagram-a Z notation based approach. In: *2006 International Conference on Computing & Informatics*. June 2006. IEEE. pp.1–6. [Online]. Available at: doi:10.1109/ICOCI.2006.5276507.

Shehab, E. M., Sharp, M. W., Supramaniam, L. and Spedding, T. A. (2012). Enterprise resource planning. *Business Process Management Journal*, 10 (4), pp.359–386. [Online]. Available at: doi:10.1108/14637150410548056.

Softwareshortlist. (2015). *Gartner's mid-market ERP Magic Quadrant: Which vendors shine? | A free ERP article from Software Shortlist*. [Online]. Available at: http://www.softwareshortlist.com/erp/articles/gartners-mid-market-erp-magic-quadrant-which-vendors-shine/.

Sommerville, I. (2016). *Software engineering (10th edition)*. Addison-Wesley Publishing Company.

Spichkova, M. (2012a). Human factors of formal methods. In: *Proceedings of the IADIS International Conference Interfaces and Human Computer Interaction 2012, IHCI 2012, Proceedings of the IADIS International Conference Game and Entertainment Technologies 2012*. 2012. Garching, Germany . pp.307–310.

Spichkova, M. (2012b). Human factors of formal methods. In: *Proceedings of the IADIS International Conference Interfaces and Human Computer Interaction 2012, IHCI 2012, Proceedings of the IADIS International Conference Game and Entertainment Technologies 2012*. 2012.

Spivey, J. M. (2010). An introduction to Z and formal specifications. *Software Engineering*

Journal, 4 (1), p.40. [Online]. Available at: doi:10.1049/sej.1989.0006.

Srihasha, A. V and Reddy, A. R. M. (2015). *Modest Formalization of Software Design Patterns*. IJLRET.

Steyn, P. S. and Van Der Poll, J. A. (2007). Validating Reasoning Heuristics Using Next-Generation Theorem-Provers. In: *Proceedings of the 5th International Workshop on Modelling, Simulation, Verification and Validation of Enterprise Information Systems*. 2007. SciTePress - Science and and Technology Publications. pp.43–52. [Online]. Available at: doi:10.5220/0002425900430052.

Subramoniam, S., Nizar, H. M., Krishnankutty, K. V. and Gopalakrishnan, N. K. (2009). ERP II: Next generation ERP. *Riyadh Community College*. [Online]. Available at: http://repository.ksu.edu.sa/jspui/handle/123456789/6421.

Suryalena. (2013). Enterprise resource planning (erp). *Jurnal Aplikasi Bisnis*, 3, pp.145–154.

Suryn, W. (2014). *Software Quality Engineering*. Suryn, W. (Ed). Hoboken, NJ, USA : John Wiley & Sons, Inc. [Online]. Available at: doi:10.1002/9781118830208.

Tomer, A. and Schach, S. R. (2002). The evolution tree: a maintenance-oriented software development model. In: *Proceedings of the Fourth European Conference on Software Maintenance and Reengineering*. 2002. IEEE Comput. Soc. pp.209–214. [Online]. Available at: doi:10.1109/CSMR.2000.827329.

Toyn, I. and McDermid, J. A. (1995). CADi: An architecture for Z tools and its implementation. *Software: Practice and Experience*, 25 (3), pp.305–330. [Online]. Available at: doi:10.1002/spe.4380250306.

Tretmans, J. and Belinfante, A. (1999). *Automatic Testing with Formal Methods*. Centre for Telematics and Information Technology (CTIT).

Van der Poll, J. A. (2010). Formal methods in software development: A road less travelled. *South African Computer Journal*, 45. [Online]. Available at: doi:10.18489/sacj.v45i0.33.

Van der Poll, J. A. and Kotze', P. (2005). Enhancing the Established Strategy for Constructing a Z Specification. *School of Computing,University of South Africa, 0003*, (35), pp.118–131.

Van Lamsweerde, A. (2000). Formal specification. In: *Proceedings of the conference on The future of Software engineering - ICSE '00*. 2000. New York, New York, USA : ACM Press. pp.147–159. [Online]. Available at: doi:10.1145/336512.336546.

Wing, J. M. (1990). A Specifier???s Introduction to Formal Methods. *Computer*, 23 (9), pp.8–22. [Online]. Available at: doi:10.1109/2.58215.

Woodcock, J., Larsen, P. G., Bicarregui, J. and Fitzgerald, J. (2009). Formal methods: Practice and experience. *ACM Comput. Surv.*, 41 (4), pp.1–36. [Online]. Available at: doi:10.1145/1592434.1592436.

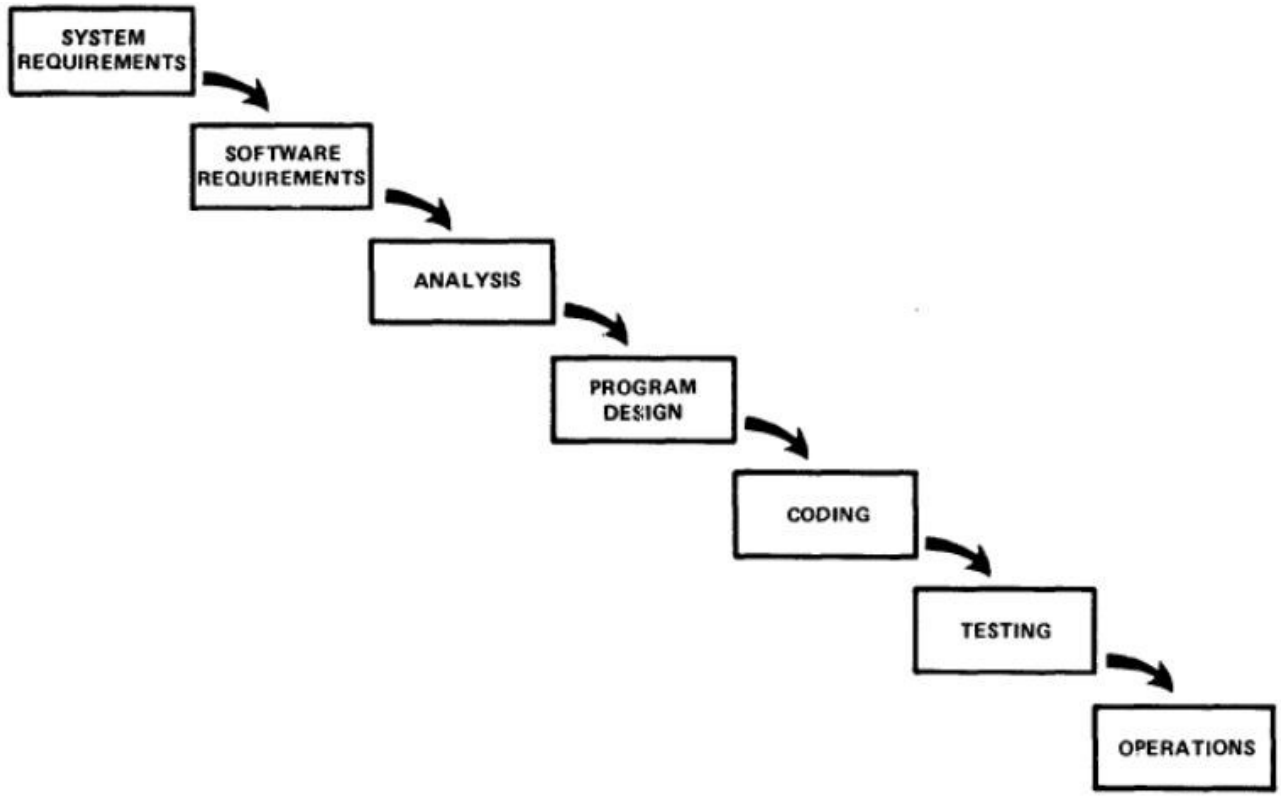
Wordsworth, J. B. (1999). *Getting the best from formal methods*. IBM United Kingdom Laboratories. [Online]. Available at: doi:10.1016/S0950-5849(99)00078-6.

Xilinx. (2012). Xilinx unveils the vivado design suite for the next decade of 'all programmable' devices. *Xcell journal*, 79 (79), p.7.

Zhang, H. (2009). An investigation of the relationships between lines of code and defects. In: *2009 IEEE International Conference on Software Maintenance*. Proceedings - Ieee International Conference on Software Maintenance. 3. September 2009. IEEE. pp.274–283. [Online]. Available at: doi:10.1109/ICSM.2009.5306304.

Appendix A. Traditional Waterfall

Traditional Waterfall development process



Source: Royce (1991), “Managing the development of large software systems”. Proc. IEEE WESCON,

Appendix B. Summary of mathematical notations

Sets

$S : \mathbb{P}X$	S is a set of X's
$x \in S$	x is a member of S
$x \notin S$	x is not a member of S
$S \subseteq T$	S is a subset of T: every member of S is also in T
$S \cup T$	the union of S and T: contains every member of S or T or both
$\{x : S \mid \ldots x \ldots\}$	the set of values of x taken from S which satisfy $\ldots x \ldots$
$\{x : S \bullet \ldots x \ldots\}$	the set of values taken by the expression $\ldots x \ldots$ as x takes values from S
$\{x, y, z\}$	the set containing just x, y, and z
$\varnothing$	the empty set: contains no members
$\mathbb{N}$	the set of natural numbers 0, 1, 2, ...
$\mathbb{N}_1$	the set of strictly positive integers 1, 2, ...
$m..n$	the set of integers from m up to n

Functions

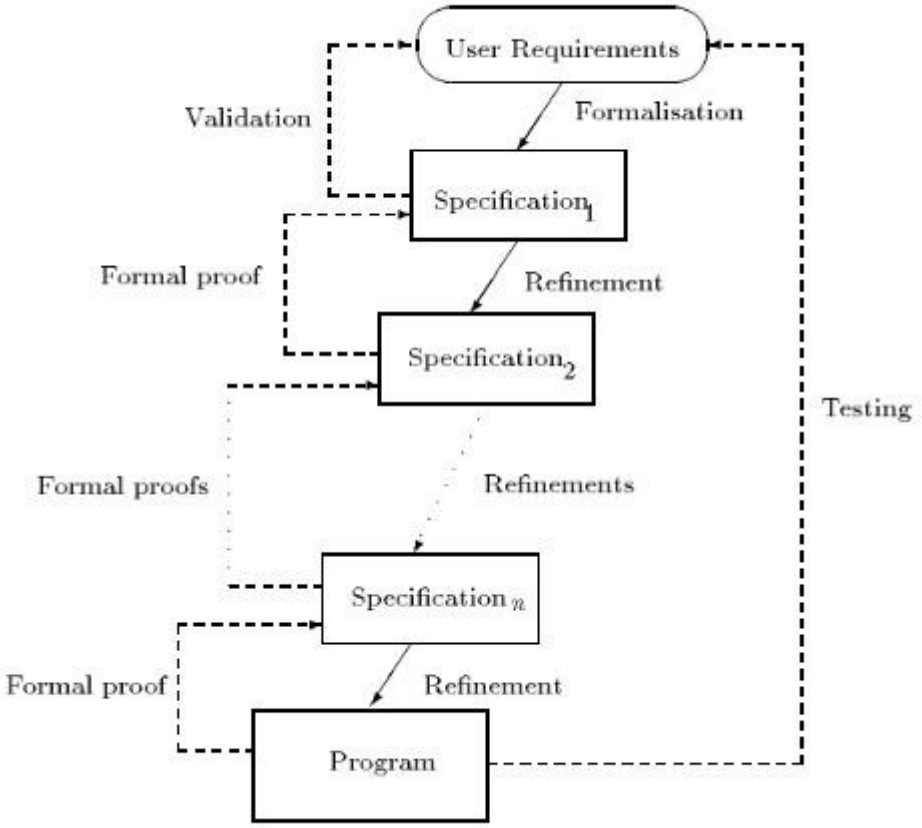
$f : X \nrightarrow Y$	f is a *partial* function from X to Y
$f(x)$	the value of f at argument x
dom f	the domain of f: the set of values of x for which $f(x)$ is defined
$f : X \rightarrow Y$	f is a total function from X to Y: dom $f = X$
$f \oplus \{x \mapsto y\}$	a function which agrees with f, except that x is mapped to y
$f \oplus g$	a function which agrees with f except inside the domain of g, where it agrees with g
$\varnothing$	the empty function: it has domain $\varnothing$

Logic

$P \wedge Q$	both P and Q are true
$P \vee Q$	P or Q or both are true
$P \Rightarrow Q$	if P is true, so is Q
$P \Leftrightarrow Q$	P is true if and only if Q is true
$\forall x : S \bullet \ldots x \ldots$	all x in the set S satisfy $\ldots x \ldots$
$\exists x : S \bullet \ldots x \ldots$	some x in the set S satisfies $\ldots x \ldots$

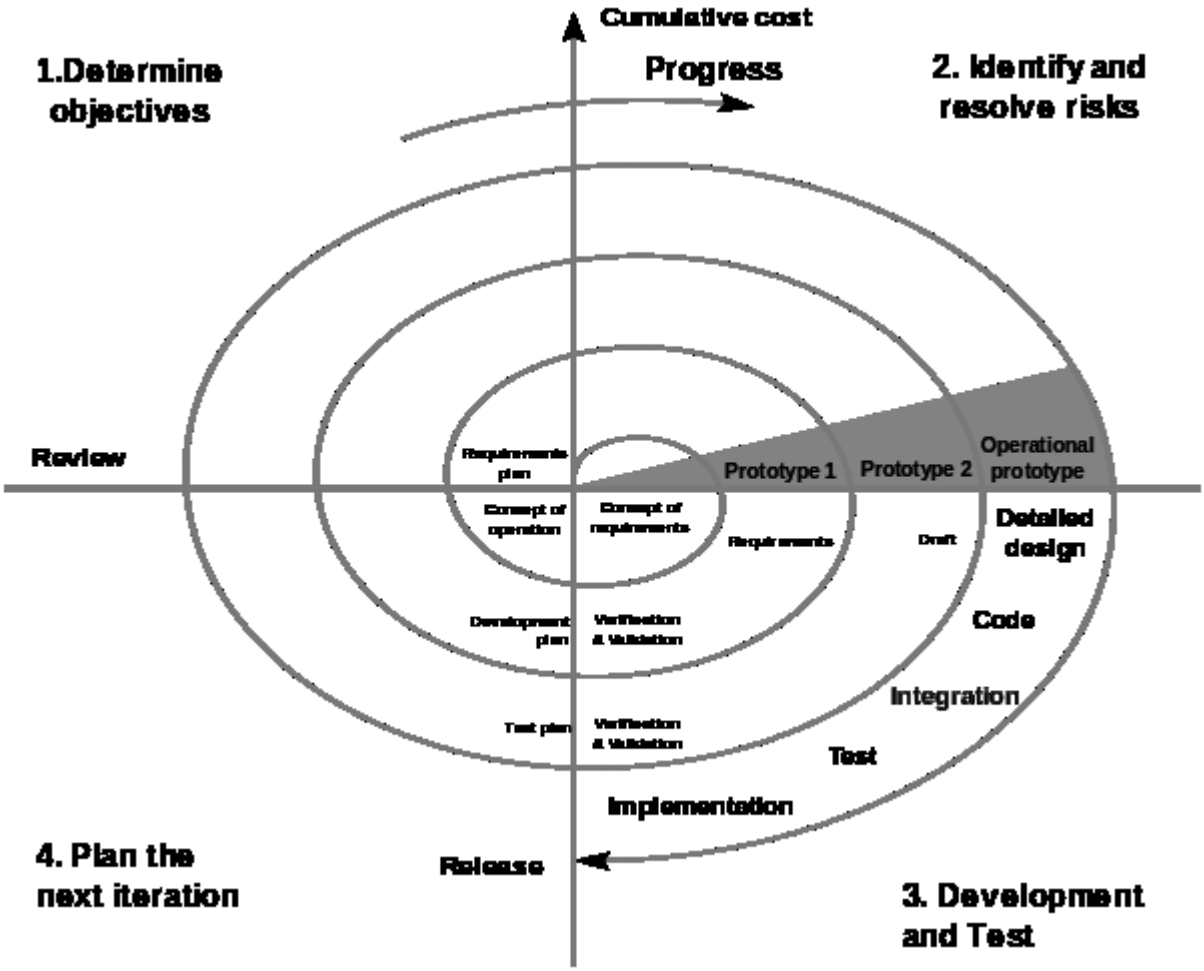
Appendix C. Software development processes

C1: Formal Methods Development Process



The process of software development using Formal Methods. Adopted from Liu and Adams (1995).

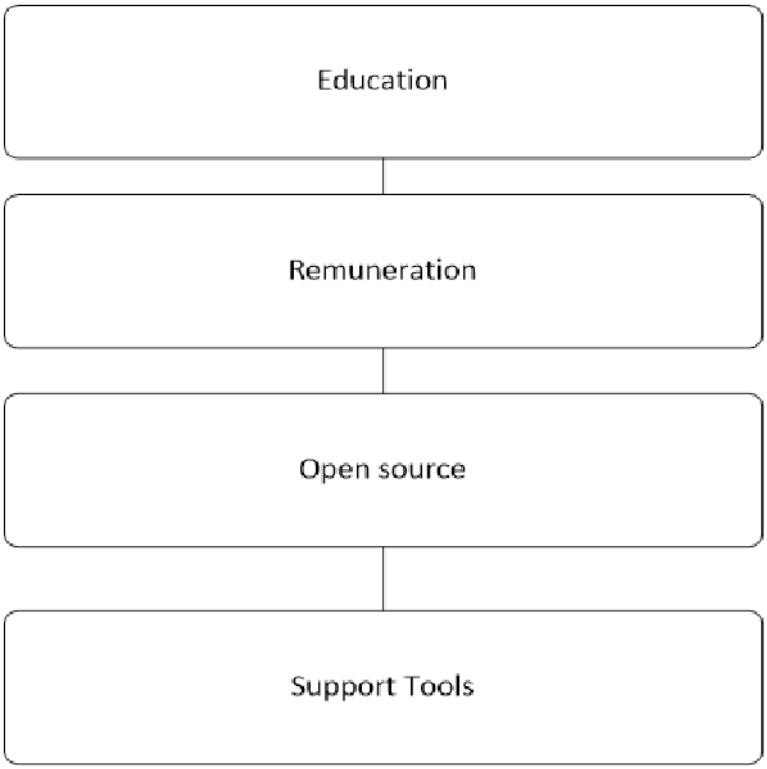
C2: Spiral Model



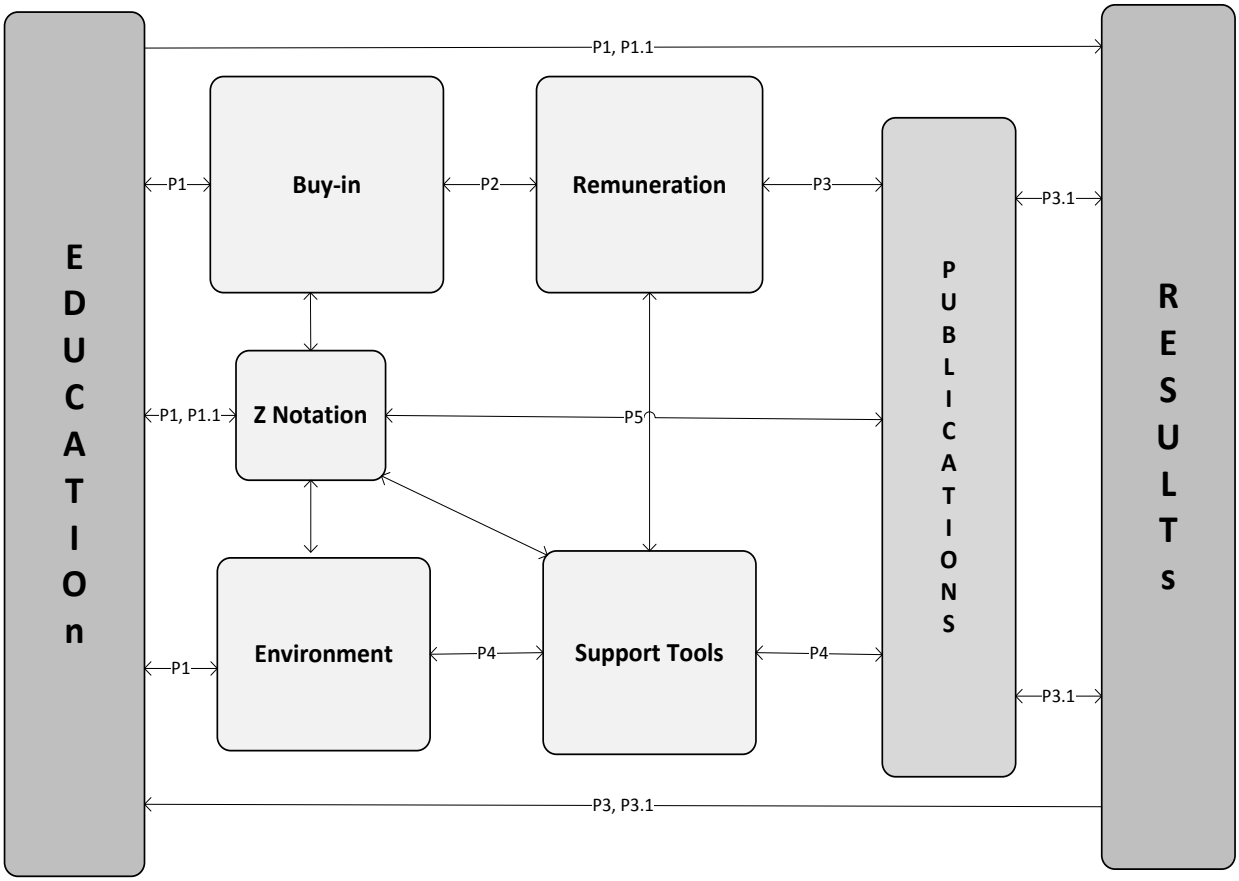
Spiral model (Pagliari, 2007)

Appendix D. Framework

D1: Preliminary Framework



D2: Adoption Framework



Appendix E. Z Specification

Z Specification Purchasing Module

E.1 Basic Types

[STRING, AMOUNT, DATE]

[USER, PRODUCT, ORDER, ITEM, CUSTOMER]

STATUS:= pending| cancelled| processed

E.2 USER

User

users: ℙ *USER*
userName: *USER* ⤔ *STRING*
userPassword: *USER* ⤔ *VARCHAR*

---

dom *userName* = *users*
dom userPassword = *users*

E.3 Log-in

Log-in

Δ*User*
username?, *password*?: *User*
r! : *RESULT*

If username? ↦ *password*? ∈ *User*
r! = *Success*
Else
r! = *Failed*

E.4 Product

Product

products: $\mathbb{P}$ *PRODUCT*
prodName: *PRODUCT* ⤔ *STRING*
prodPrice: *PRODUCT* ⇸ *AMOUNT*
prodQuantity: *PRODUCT* ⇸ $\mathbb{N}$

---

dom *prodName* = *products*
dom *prodPrice* = *products*
dom *proQuantity* = *products*

E.5 Customer

Customer

customers: $\mathbb{P}$ *CUSTOMER*
custAddress: *CUSTOMER* ⇸ *STRING*
custPhone: *CUSTOMER* ⇸ *STRING*

---

dom *custAddress* = *customers*
dom *custPhone* = *customers*

E.6 Order

Order

order: $\mathbb{P}$ *ORDER*
orderDate: *ORDR* ⇸*DATE*
orderStatus: *ORDR* ⇸*STATUS*
orderCustomer: *ORDR* ⇸*CUSTOMER*

---

dom *orderDate* = *order*
dom *orderStatus* = *order*
dom *orderCustomer* = *order*

E.7 Create Product Operation

Create Product Operation

CreateProduct

$\Delta Product$
$prduct? : PRODUCT$
$nme? : STRING$
$prce? : AMOUNT$
$qntity? : \mathbb{N}$

$prduct? \notin products$
$products' = products \cup \{prduct?\}$
$prodName' = prodName \cup \{prduct? \mapsto nme?\}$
$prodPrice' = prodPrice \cup \{prduct? \mapsto prce?\}$
$prodQuantity' = prodQuantity \cup \{prduct? \mapsto qntity?\}$

E.8 Create Order

CreateOrder

$\Delta Order$
$date? : DATE$
$customer? : CUSTOMER$
$ordr! : ORDER$

$order! \notin orders$
$ordr' = orders \cup \{ordr!\}$
$orderDate' = orderDate \cup \{ordr! \mapsto date?\}$
$orderStatus' = orderStatus \cup \{ordr! \mapsto pending\}$
$orderCustomer' = orderCustomer \{ordr! \mapsto customer?\}$

E.9 Create Customer

CreateCustomer

$\Delta Customer$
customer?: *CUSTOMER*
address?: *STRING*
phone?: *STRING*

$customer? \notin customers$
$customers' = customers \cup \{customer?\}$
$custAddress' = custAddress \cup \{customer? \mapsto address?\}$
$custPhone' = custPhone \cup \{customer? \mapsto phone?\}$

E.10 Process Order

ProcessOrder

$\Delta Order$
$\Delta Product$
$\Xi Customer$ (* *Yet, a real-life system would maintain some customer information* *)
product? : *PRODUCT*
ordr?: *ORDER*

$ordr? \in orders \land product? \in products$

(* *Valid pending order and product stock available* *)
$orderStatus(ordr?) = pending \land prodQuantity(product?) > 0$

(* *Components that remain invariant* *)
$orders' = orders$
$orderDate' = orderDate$
$orderCustomer' = orderCustomer$
$products' = products$
$prodName' = prodName$
$prodPrice' = prodPrice$

(* *New status of order* *)
$orderStatus' = orderStatus \oplus \{ordr? \mapsto processed\}$

(* *New quantity of product* *)
$prodQuantity' = prodQuantity \oplus$
$\{(product? \mapsto prodQuantity(product?)) - orderQuantity(ordr?)\}$

E.11 Update Product

UpdateProduct

$\Delta Product$
product?: *PRODUCT*
nme?: *STRING*
prce?: *AMOUNT*
qntity?: $\mathbb{N}$

---

product? $\in$ *products*
prodName' = *prodName* $\oplus$ {*product*? $\mapsto$ *nme*?}
prodPrice' = *prodPrice* $\oplus$ {*product*? $\mapsto$ *prce*?}
(* *Abstracting away from order-product relationship in schema ProcessOrder above* *)
prodQuantity' = *prodQuantityy* $\oplus$ {*product*? $\mapsto$ *qntity*?}

E.12 Delete Product

DeleteProduct

$\Delta Product$
product?: *PRODUCT*

---

product? $\in$ *products*
products' = *products* $\setminus$ {*product*?}
prodName' = {*product*?} $\lhd$ *prodName*
prodPrice' = {*product*?} $\lhd$ *prodPrice*
prodQuantity' = {*product*?} $\lhd$ *prodQuantity*

E.13 Cancel Order

CancelOrder

$\Delta Order'$
ordr?: *ORDER*

---

ordr? $\in$ *orders*
orderStatus(*ordr*?) = *pending*
orderDate' = *orderDate*
orderStatus' = *orderStatus* $\oplus$ {*ordr*? $\mapsto$ *cancelled*}
orderCustomer' = *orderCustomer*

E.14 Enquiry Operation

SelectProductsBelowThreshold

Ξ*Product*
quantity?: $\mathbb{N}$
products!: $\mathbb{P}$ *PRODUCT*

---

products!= {*p*: *products* | *prodQuantity*(*p*) < *quantity*?}

E.15 Operation success

Success

result!: *REPORT*

---

result!: = *success*

E.16 Error condition

ProductAlreadyExists

Ξ*Product*
prduct?: *PRODUCT*
result!: *REPORT*

---

prduct? $\in$ *products*
result!: = *product_already_exists*

E.17 Report

RCreateProduct

$\Delta Product$
product?:*PRODUCT*
name?: *STRING*
price?: *AMOUNT*
quantity?: $\mathbb{N}$
results! : *REPORT*

(*product*? $\notin$ *known* $\land$
Product ' = *product* $\cup$ {*name*? $\mapsto$ *price*? } $\land$
result! = *ok*) $\lor$
(*name*? $\in$ *known* $\land$
product' = *product* $\land$
result! = *already_ known*)

E.18 System

System

User
Product
Order
Customer

Appendix F. Ethical Clearance

The following ethical clearance certificate was obtained as part of non-human subjects research.

UNISA university of south africa

UNISA COLLEGE OF SCIENCE, ENGINEERING AND TECHNOLOGY'S (CSET) RESEARCH AND ETHICS COMMITTEE

10 September 2019

Ref #: 063/APN/2019/CSET_SOC
Name: Mr Aifheli Peter Nemathaga
Student #: 51843331

Dear Mr Aifheli Peter Nemathaga

Decision: Ethics Approval for 3 years
(No Humans involved)

Researchers: Mr Aifheli Peter Nemathaga, 51843331@mylife.unisa.ac.za, +27 78 539 4997, +21 84 174 4714

Project Leader(s): Prof JA van der Poll, vdpolja@unisa.ac.za, +27 11 652 0316

Working title of Research

Formal Methods Adoption in the Commercial World

Qualification: MSc in Computing

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 10 September 2019 to 10 September 2022.

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 be requested if there are substantial changes from the existing proposal, especially if those changes affect any of the study-related risks for the research participants.

University of South Africa
Preller Street, Muckleneuk Ridge, City of Tshwane
PO Box 392 UNISA 0003 South Africa
Telephone: +27 12 429 3111 Facsimile: +27 12 429 4150
www.unisa.ac.za

3. The researcher(s) will conduct the study according to the methods and procedures set out in the approved application.
4. Any changes that can affect the study-related risks for the research participants, particularly in terms of assurances made with regards to the protection of participants' privacy and the confidentiality of the data, should be reported to the Committee in writing, accompanied by a progress report.
5. The researcher will ensure that the research project adheres to any applicable national legislation, professional codes of conduct, institutional guidelines and scientific standards relevant to the specific field of study. Adherence to the following South African legislation is important, if applicable: Protection of Personal Information Act, no 4 of 2013; Children's act no 38 of 2005 and the National Health Act, no 61 of 2003.
6. Only de-identified research data may be used for secondary research purposes in future on condition that the research objectives are similar to those of the original research.
7. Submission of a completed research ethics progress report will constitute an application for renewal of Ethics Research Committee approval.

Note:

The reference number 063/APN/2019/CSET_SOC *should be clearly indicated on all forms of communication with the intended research participants, as well as with the Unisa College of Science, Engineering and Technology's (CSET) Research and Ethics Committee.*

Yours sincerely

_____

Dr. B Chimbo

Chair: Ethics Sub-Committee SoC, College of Science, Engineering and Technology (CSET)

2019/09/12

_____

Dr MG Katumba

Director: School of Computing, CSET

_____

Prof B. Mamba

Executive Dean: CSET

University of South Africa
Preller Street, Muckleneuk Ridge, City of Tshwane
PO Box 392 UNISA 0003 South Africa
Telephone: +27 12 429 3111 Facsimile: +27 12 429 4150
www.unisa.ac.za

Appendix G. Permission to Submit

Dear Student

I acknowledge receipt of your recent correspondence and have noted that you intend submitting your research output for examination. **Regarding submission dates the following rules apply:**

- If submission takes place after 15 June, the successful student might only graduate in Autumn of the following year.
- If submission takes place after 15 November, the successful student might only graduate in Spring of the following year.
- **If submission takes place after the 31 January, the successful student will graduate in Spring, and will have to re-register and pay the full tuition fees.**
- If you are not currently a registered student, examination will be delayed until proof of registration had been submitted by you

Your request for submission has been referred, *inter alia*, for the appointment of a panel of examiners and it could take some time. You will be informed of approval of submission in due course.

In order to avoid any possible delay in having your dissertation examined, kindly ensure that you comply with all the requirements regarding the following:

- the dissertation and the submission thereof,
- the **exact wording of the approved title in the correct format** on the title page as indicated in the *example* attached hereto
- the **limitation** of **the summary to a maximum of 150 words**, as well as at least **ten key terms** listed at the end of the summary
- the submission of a declaration, **signed and dated** by you, including your **student number** on the statement, indicating that the dissertation is your own work

Yours faithfully

for THE EXECUTIVE DEAN: COLLEGE OF GRADUATE STUDIES

Appendix H. Language Editing Certificate

This dissertation has been professionally language edited as indicated below.

GENEVIEVE WOOD
P.O. BOX 511 WITS 2050 | 0616387159

EDITING CERTIFICATE

LANGUAGE EDITING SERVICES

Date: 2019/12/12

This serves to confirm that the document entitled:

Formal Methods Adoption in the Commercial World

by

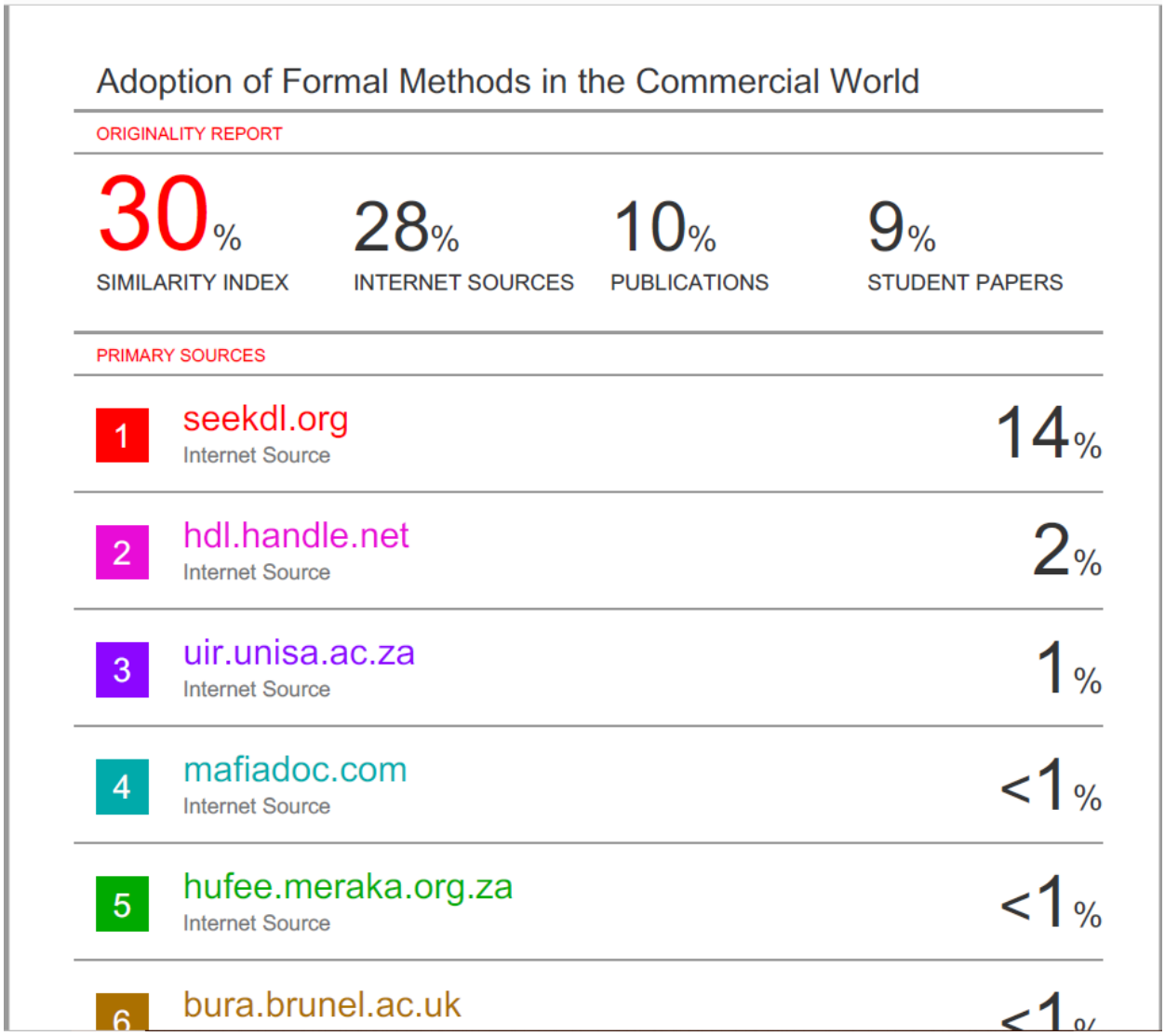
Aifheli Nemathaga

has been language edited on behalf of its author, with recommendations for improvement.

Genevieve Wood
PhD candidate
Wits University

Appendix I. Turnitin Report

NOTE: A research paper by Nemathaga, A. and van der Poll, J. A. (2019) Adoption of Formal Methods in the Commercial World published as a Master's degree requirement appears in the *Eighth International Conference on Advances in Computing, Communication and Information Technology (CCIT 2019)*. This resulted in the Turnitin report similarity index of 30%. The work is by the researcher (14%) and was not plagiarised in any way. The University discourages the removal of previously published works by candidates from the Turnitin repository. The full paper appears in *Appendix J.*



Adoption of Formal Methods in the Commercial World

ORIGINALITY REPORT

30%	28%	10%	9%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	seekdl.org Internet Source	14%
2	hdl.handle.net Internet Source	2%
3	uir.unisa.ac.za Internet Source	1%
4	mafiadoc.com Internet Source	<1%
5	hufee.meraka.org.za Internet Source	<1%
6	bura.brunel.ac.uk	<1%

Appendix J. Adoption of Formal Methods in the Commercial World

The following publication emanated from the research described in this dissertation.

- Nemathaga, A. and van der Poll, J. A. (2019) 'Adoption of Formal Methods in the Commercial World', in *Eighth International Conference on Advances in Computing, Communication and Information Technology CCIT*. Birmingham City University, Birmingham, United Kingdom: Institute of Research Engineers and Doctors., pp. 75–84. doi: 10.15224/978-1-63248-169-6-12.

Proc. of the Eighth Intl. Conf. on Advances in Computing, Communication and Information Technology - CCIT 2019

ISBN: 978-1-63248-169-6 DOI : 10.15224/978-1-63248-169-6-12

Adoption of Formal Methods in the Commercial World

[Aifheli Nemathaga and John Andrew van der Poll]

Abstract — **There have been numerous studies on formal methods but yet there is diminutive utilization of formal methods in the commercial world. This can be attributed to many factors such as that few specialists know how to use Formal Methods (FMs) and also the use of mathematical notation gives a perception that formal methods are hard. FMs have been used in the software development world since 1940 during the earliest stage of computer development. To date there is a slow adoption of FMs and they are used mostly in mission critical projects such as in the military and aviation. In this paper we focus on how to increase the pace of FMs adoption in the commercial world. As part of this work a framework is established to facilitate the use of FMs in the commercial world or commercial systems. A mini ERP system specification is presented in both an informal technique and a formal notation to demonstrate how a formal specification can be derived from informal specification guided by the Enhanced Established Strategy for formal specification.**

Keywords — **Commercial system, ERP, First-order logic, Formal Methods (FMs), Formal Specification, Formal Verification, Set theory, TLA+, Z, Zermelo-Fraenkel**

I. Introduction

This paper investigates the feasibility of utilising Formal Methods (FMs) within commercial software development. In addition to the findings it defines and develops a framework to facilitate the use of FMs in commercial software development. The paper places the focus on ERP systems and a small formal methods specification is written specifying requirements for an ERP system.

This paper is organized as follows: Section II provides an overview and context of formal methods. Section III of the paper is about formal methods adoption further explaining A – the reason for slow adoption and B, the differences between FMs and natural language/prose. Section IV is about formal methods in practise which leads to Section V which in turn gives practical examples of formal methods in the commercial world. Section VI presents a formal methods adoption framework and Section VII concludes the paper and gives directions for future work in this area.

Aifheli Nemathaga
School of Computing (SoC), Florida Campus
University of South Africa
South Africa

John Andrew van der Poll
Graduate School of Business Leadership (SBL), Midrand Campus
University of South Africa
South Africa

II. FMs Overview and Context

The advancement of hardware during the past 30 years has led to the development of large and complex systems. The growing technologies range from mobile devices, industrial machinery and automobiles. These systems require fast processing in order for hardware and software to work together to perform complex tasks [44]. The lines of code have increased from a couple of lines to 40 million lines in software and it is still increasing. As these systems grow designers and engineers face many challenges. These systems are designed, enhanced and modified often during their lifetime. Software development is time consuming and a costly process, and research has shown most software do not meet users' needs and is delivered out of their respective budgets [6]. This also applies to ERP systems, that is ERP project implementations are mostly unsuccessful or implemented out of timelines and with higher costs [39]. Consequently, many software development techniques have been developed to try to overcome these challenges.

Formal methods have shown to be one of the promising techniques to potentially overcome some of the above challenges. There are numerous benefits in using formal methods, e.g. they have been shown to reduce the number of defects in software development [1]. In the software development world there is always a search to find better ways of developing software that are free from errors and delivered within timelines and on budget. This led to the development of various frameworks and methodologies of software development. The most famous and widely used is the traditional waterfall methodology which proposes that software has to be developed using a stepwise approach, i.e. requirements, design, implementation, verification and maintenance [31]. Each stage must be finalised prior to starting the next. Waterfall is one of the oldest models still used today [28]. Yet, many of the waterfall projects are delivered out of budget, with many defects and the end product usually does not present the real needs of the user [29]. There is an increased uptake of the Agile methodology in the commercial world; software is developed in increments and in rapid cycles. Agile's main objective is to deliver value to customer by means of working software [4]. An agile methodology is guided by a manifesto which defines the principles that must be followed when using Agile. That said, Agile has many disadvantages such as a lack of documentation and the project can easily be taken out of track if a customer's requirements are not well understood. Given the aforementioned, formal methods can plausibly be incorporated during any stage or phase of the SDLC and has proven to reduce the error count [14].

III. Formal Methods Adoption

Software testing has traditionally been the only technique that has been used and is still used to find defects. Yet, code testing is not an effective way of finding subtle

Proc. of the Eighth Intl. Conf. on Advances in Computing, Communication and Information Technology - CCIT 2019

ISBN: 978-1-63248-169-6 DOI : 10.15224/978-1-63248-169-6-12

Adoption of Formal Methods in the Commercial World

[Aifheli Nemathaga and John Andrew van der Poll]

Abstract — **There have been numerous studies on formal methods but yet there is diminutive utilization of formal methods in the commercial world. This can be attributed to many factors such as that few specialists know how to use Formal Methods (FMs) and also the use of mathematical notation gives a perception that formal methods are hard. FMs have been used in the software development world since 1940 during the earliest stage of computer development. To date there is a slow adoption of FMs and they are used mostly in mission critical projects such as in the military and aviation. In this paper we focus on how to increase the pace of FMs adoption in the commercial world. As part of this work a framework is established to facilitate the use of FMs in the commercial world or commercial systems. A mini ERP system specification is presented in both an informal technique and a formal notation to demonstrate how a formal specification can be derived from informal specification guided by the Enhanced Established Strategy for formal specification.**

Keywords — **Commercial system, ERP, First-order logic, Formal Methods (FMs), Formal Specification, Formal Verification, Set theory, TLA+, Z, Zermelo-Fraenkel**

I. Introduction

This paper investigates the feasibility of utilising Formal Methods (FMs) within commercial software development. In addition to the findings it defines and develops a framework to facilitate the use of FMs in commercial software development. The paper places the focus on ERP systems and a small formal methods specification is written specifying requirements for an ERP system.

This paper is organized as follows: Section II provides an overview and context of formal methods. Section III of the paper is about formal methods adoption further explaining A – the reason for slow adoption and B, the differences between FMs and natural language/prose. Section IV is about formal methods in practise which leads to Section V which in turn gives practical examples of formal methods in the commercial world. Section VI presents a formal methods adoption framework and Section VII concludes the paper and gives directions for future work in this area.

Aifheli Nemathaga
School of Computing (SoC), Florida Campus
University of South Africa
South Africa

John Andrew van der Poll
Graduate School of Business Leadership (SBL), Midrand Campus
University of South Africa
South Africa

II. FMs Overview and Context

The advancement of hardware during the past 30 years has led to the development of large and complex systems. The growing technologies range from mobile devices, industrial machinery and automobiles. These systems require fast processing in order for hardware and software to work together to perform complex tasks [44]. The lines of code have increased from a couple of lines to 40 million lines in software and it is still increasing. As these systems grow designers and engineers face many challenges. These systems are designed, enhanced and modified often during their lifetime. Software development is time consuming and a costly process, and research has shown most software do not meet users' needs and is delivered out of their respective budgets [6]. This also applies to ERP systems, that is ERP project implementations are mostly unsuccessful or implemented out of timelines and with higher costs [39]. Consequently, many software development techniques have been developed to try to overcome these challenges.

Formal methods have shown to be one of the promising techniques to potentially overcome some of the above challenges. There are numerous benefits in using formal methods, e.g. they have been shown to reduce the number of defects in software development [1]. In the software development world there is always a search to find better ways of developing software that are free from errors and delivered within timelines and on budget. This led to the development of various frameworks and methodologies of software development. The most famous and widely used is the traditional waterfall methodology which proposes that software has to be developed using a stepwise approach, i.e. requirements, design, implementation, verification and maintenance [31]. Each stage must be finalised prior to starting the next. Waterfall is one of the oldest models still used today [28]. Yet, many of the waterfall projects are delivered out of budget, with many defects and the end product usually does not present the real needs of the user [29]. There is an increased uptake of the Agile methodology in the commercial world; software is developed in increments and in rapid cycles. Agile's main objective is to deliver value to customer by means of working software [4]. An agile methodology is guided by a manifesto which defines the principles that must be followed when using Agile. That said, Agile has many disadvantages such as a lack of documentation and the project can easily be taken out of track if a customer's requirements are not well understood. Given the aforementioned, formal methods can plausibly be incorporated during any stage or phase of the SDLC and has proven to reduce the error count [14].

III. Formal Methods Adoption

Software testing has traditionally been the only technique that has been used and is still used to find defects. Yet, code testing is not an effective way of finding subtle

Proc. of the Eighth Intl. Conf. on Advances in Computing, Communication and Information Technology - CCIT 2019

ISBN: 978-1-63248-169-6 DOI : 10.15224/978-1-63248-169-6-12

bugs/error in the design. The use of formal methods helps to reduce errors early in software development, thereby saving on the cost of software projects. Formal methods are categorised mainly in two groups, namely, (1) Pure mathematics which is challenging and is mostly not used in the real world, and (2) Software engineering which focuses on creating high quality software [42].

Formal methods use mathematics to analyze and verify models at any stage of the software development process [45]. One of the most significant parts of the development process is to understand the needs of the users. According to George and Vaughn [14] formal methods are useful when gathering, articulating, and representing requirements. This then assists the developer in developing a system that meets a user's needs.

Another type of formal methods is the state-based method, which involves the creation of state machine specifications, simulation proofs and abstract functions. During development formal methods are used to verify code by attempting to prove theorems (discharging proof obligations) about the proposed system. Some tools used for formal methods can automatically generate compilable code e.g. the B-method. The clarity, completeness and consistency of a formal specification facilitate the derivation of test cases [41]. As part of this paper a formal specification will be documented using the Z notation which is a formal language as indicated in Figure 1.

	Sequential	Concurrent
Algebric	Larch (Guttag, et al., 1993) OBJ (Futatsugi, el al., 1985)	Lotos (Bolognesi and Brinksma, 1987),
Model-based	Z (Spivey, 1992) VDM (Jones, 1980) B (Wordsworth, 1996)	CSP (Hoere, 1985) Petri Nets (Peterson, 1981)

Figure 1. Formal Specification Languages [34].

A. *Reasons for Slow Adoption*

Many software development institutions don't consider it to be cost-effective to incorporate FMs in their software development processes [34]. Some of the stumbling blocks in the use of FMs in the commercial world is the perception that requirements formalization is difficult and the creation of a formal specification as part of the formal methods process is error prone and time consuming [3]. Formal methods are based on mathematical notations, which are perceived as being hard, but in reality the notation can readily be mastered and used. For example, it is easier to learn such notation than learning a new programming language [7].

Another reason for the slow adoption of FMs is that most engineers also view FMs to be a mechanism that is practically hard to understand and utilise [35]. In the same vein, the commercial world or businesses are of the view that using FMs will increase the cost of system development due the level of training required. Education plays a major role to individuals developing and designing systems. In addition also management need to be educated if they are to successfully apply formal methods within their organizations [45].

The above ideas lead to the following proposition to be used in the construction of our formal-methods adoption framework:

Proposition (Prop) 1: *Education plays a major role in formal methods adoption. This includes educating from the high school level to the university level as well as organisational training in formal methods. Such education plays a pivotal role in the adoption framework.*

As more software development processes gain popularity e.g. the Agile methodology, there is a view that formal methods do not support other software development processes. Yet, formal methods can be beneficial in every step of the software development life cycle as they assist in alleviating unclear and unrealistic requirements at the start of the development process, leading to the production of a high quality product with fewer defects [11].

The lack of easy step-by-step guidelines on how to use formal methods also contributes to the slow adoption. Many developers view formal methods as being limited to academic projects for tertiary education. Bowen & Hinchey [8] postulated that tools, standards, and education would make or break commercial adoption, while some observed differences among academics who view FMs as "inevitable".

Most traditional software development techniques are established and proper standards have been set. Tools supporting those techniques are widely accepted and used in business [35]. On the other hand, FMs appear to have inadequate tool support. Some formal methods tools do not work suitably with development/programming tools. Formal methods tools are also not seen as facilitating the user experience (UX) [44].

When compared to traditional techniques there are many certifications that one can acquire and many institutions offer training around such techniques. The study done by Davis [10] shows that formal methods adoption may also be attributed to certification authorities not having enough education on how to appraise FMs artefacts and they are not highly informed of formal methods benefits and underlying techniques.

The above discussion leads to further insight into Proposition 1 as follows:

Prop 1.1: *In addition to the above proposition formal certificates and diplomas in formal methods should be created and awarded to those who qualify. Certification authorities should be well informed about the benefits of formal methods.*

Usually when developing a system for clients, users review and sign off the requirements specification i.e. the Business Requirement specification (BRS) or Functional Requirement Specification (FRS). The reviews are for validation and ensuring all user requirements are included in the specification [18]. The specification can then be used to bill an external client; for an internal client an agreement could confirm that the stated requirements will be developed. Clients find it hard to review formal specifications due to the mathematical notations used, resulting in project delays.

Proc. of the Eighth Intl. Conf. on Advances in Computing, Communication and Information Technology - CCIT 2019

ISBN: 978-1-63248-169-6 DOI : 10.15224/978-1-63248-169-6-12

There is also a psychological and human resource factor with the slow adoption of FMs in business. Within the organization or business some people just do not like formalism; the same applies to formal methods as some engineers, especially those who are already in an agile environment, will be more reluctant to use FMs [35]. In the commercial world the development of some projects are relatively fast, so there is little time to conduct a proper formal analysis. In today's world individuals change positions frequently, for example from a software engineer to a manager or even changing companies. This results in having to upskill a new employee which is time consuming.

Consequently we arrive at our next proposition:

Prop 2: *Buy-in from all the business stakeholders is necessary for FMs adoption. Getting Top level management to agree and accept the use of formal methods may well result in the whole organisation adopting formal methods.*

As indicated, there are numerous misunderstandings with formal methods, leading to slow adoption in business. Businesses view formal methods as a technique that places too much emphasis on the theory, rather than real world applications. Another huge misconception is that if FMs are used, then there is no need for testing; this relates to one of the 7 FMs myths – the use of a formal method guarantees that the resulting software or system is perfect [20].

There is a huge gap between the real world and formalism, namely, transforming clients' requirements from informal requirements to formal requirements requires serious clarification of the problem [24]. Generally, there is no widely accepted principle or guidance of eliciting a client's requirements and how to specify them using a formal specification language [20].

Sommerville [34] indicated four (4) reasons of why there is a slow adoption of FMs in the commercial world:

1. The utilization of other system engineering techniques, e.g. configuration management and structured techniques have improved software quality.
2. Lately, software is developed and delivered fast. The main focus is time to market (TTM) rather than quality; in some instances customers will accept software with some errors so long as it can be delivered rapidly. Rapid software delivery does not work well with formal methods.
3. The narrow scope of formal methods often does not cater for user interface design and user interaction.
4. Developing a formal specification for a system upgrade becomes a time consuming and costly process, aspects which the commercial world are unlikely to compromise on.

The above leads to the following proposition:

Prop 3: *Widely accepted principles and guidelines on FMs can improve the adoption thereof. Practical, real world examples of FMs successes and failures must be published in the software engineering and management communities.*

B. *Differences between formal and informal (natural language) specifications*

TABLE I. DIFFERENCES BETWEEN FORMAL AND INFORMAL SPECIFICATIONS [19].

Informal (natural language)	Formal Methods
Each stakeholder has his/her own interpretation of the requirements.	There is generally a complete and broad view of system requirements.
More errors present, and if not corrected can result in high project costs.	Fewer errors and omissions in the specification document.
Uses a combination of graphics and semiformal notations.	Uses mathematical notation, first-order logic and natural language prose.
Little or no use of Mathematics. General knowledge on the software engineering domain used.	Specifiers and stakeholders ought to be familiar with the mathematical notation.
Specifiers leave room for inconsistency and ambiguity.	Provides conciseness, clarity and unambiguity.
They are ideal for eliciting requirements.	Allows the software engineer to produce high quality systems.

Both formal notations and informal techniques can result in a vague understanding of the system [24]. All this depends on the software engineer or the developer understanding what to build irrespective of the language used in the specification. Depending on the specifier's willingness, it is possible to learn formal languages but it takes time and may be a costly exercise.

IV. FMs in practice

For this paper we will show how FMs are written and the chosen language for this paper is Z [36]. The commercial system that will be specified formally is an ERP system.

A brief discussion of what an ERP system is follows next.

A. *What is Enterprise Resource Planning (ERP)?*

As per Seo [32] "ERP is a software architecture that is designed in order to expedite the information flow as well as the information sharing between various departments in a company, and also to provide to decision-makers an enterprise-wide view of all information that they may need to assist them in decision-making".

From the above description it follows that Enterprise Resource Planning (ERP) systems are combined software programmes that are clustered into standard functional modules i.e. Procurement, Human resources, Finance, Contract management, Customer relationship management (CRM), etc. developed by a Vendor or in-house [26]. Usually a single database, or more generally a data warehouse, together with a unified interface across the entire enterprise is utilized [2]. Some ERP software can be purchased off the shelf and customized afterwards to meet specific customer needs. An ERP system assists business in performing their daily operations which can bring about

Proc. of the Eighth Intl. Conf. on Advances in Computing, Communication and Information Technology - CCIT 2019

ISBN: 978-1-63248-169-6 DOI : 10.15224/978-1-63248-169-6-12

substantial benefits within the organization. Despite all the benefits mentioned above, ERP project implementations are mostly unsuccessful or implemented out of timelines and with higher costs than budgeted [39].

Failure of ERP project implementation can be attributed to different factors such as unclear requirements, project managers focusing on the financial aspect of the business and neglecting other parts of the project and no proper software development process in place to manage the projects to name a few. Most of the time the success of the project is attributed to delivering the project within the timelines and on budget; developers and managers tend to forget about the users of the system and the smooth change from a previous (existing) process to the new one [16]. Most of the aforementioned failures do apply when implementing other software such as CRM, Billing systems etc. The researchers believe the use of formal methods will help alleviate many of the problems during ERP implementation in an organization.

1) ERP Modules

The ERP architecture in Figure 2 presents some of the main modules that are contained in an ERP system.

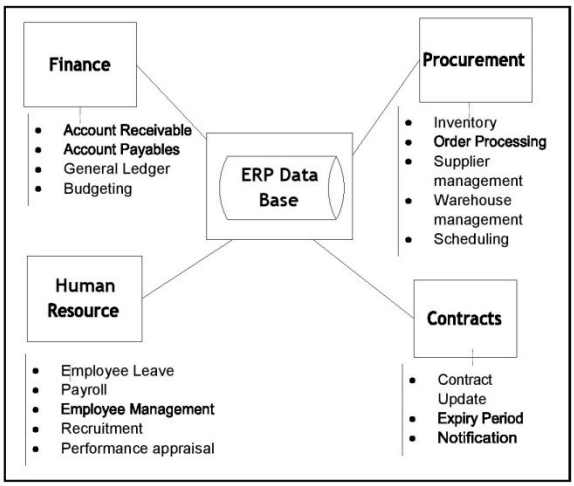


Figure 2. An ERP Architecture [22] .

The focus of this paper will be on the *procurement* module otherwise known as the purchasing module. The reason for this is that a procurement module is usually the most widely used module in an organisation [13].

The Procurement module deals with the purchasing of materials for internal use or resale within the organization. Procurement mostly have a built-in workflow which can automatically evaluate the supplier, measure the inventory at hand which is otherwise known as warehouse management. Lastly, most of the purchasing modules are integrated into invoice verification. Gao [13] calls the procurement module the "internet procurement".

B. *The Z Specification Language*

The Z specification language was established in the late 1970s at Oxford University by the Programming Research Group otherwise known as the PRG [11]. Z is constructed on a strongly typed fragment of Zermelo-Frankel (mathematical) set-theory and first-order logic; it embodies a rich notation. Using a formal specification language such as Z, software systems can be designed with little uncertainties [17]. Many formatting tools such as Latex and type-checkers for writing syntactically correct Z specifications have been developed since Z is written mostly in non-ASCII mathematical symbols (refer to the Z schemas that follow).

A Z specification comprises of schemas with narrative text in-between. A schema is an organizing unit that contains logically associated mathematical notation.

Below is the generic format of a Z schema:

$$\begin{array}{|l} \textit{SchemaName} \\ \hline \textit{Declarations} \\ \hline \textit{Predicate}_1;\ldots;\textit{Predicate}_n \\ \hline \end{array}$$

Apart from a few exceptions, a Z schema is usually divided into 2 parts as follows. Part one above the short (middle) dividing line specifies the variables (components) and their types (declarations) to be used in the specification. Part two specifies, in first-order notation, the constraints on, and relationships among the components of the specification.

1) Some of the tools that are used for a Z specification

Naturally, tool support assists a lot when developing a Z specification. One of the add-on advantages of Z over some other languages is a number of software tools that have been developed to assist with the construction of a specification [42].

- CadiZ – Computer aided design in Z by Mcdermid and Toyn [40], used to construct, type check and reason about a Z specification.
- Fuzz – Mike Spivey's commercial type checker for Z [36]. More information is available at the URL: http://spivey.oriel.ox.ac.uk/corner/Fuzz_typechecker_for_Z.
- The Community Z Tool (CZT) by Utting and Malik [25] (http://czt.sourceforge.net/) is another useful addition to the arsenal for writing Z specifications.

Using a formal notation assists in understanding how the system will operate and it allows the developer to have more choices about the design of the system [17]. The omitted parts of the specification become easy to identify and the overall quality of the specification document is enhanced.

The above discussions leads to our next proposition:

Prop4: *Tools that are readily available and up to date with the lasted technology should facilitate the adoption of FMs. Such tools should be integrated with the requirements management software and standard software programming tools e.g., MS Visual Studio.*

The development of the Z specification presented below will be guided by the Enhanced Established Strategy (E^2S)

Proc. of the Eighth Intl. Conf. on Advances in Computing, Communication and Information Technology - CCIT 2019

ISBN: 978-1-63248-169-6 DOI : 10.15224/978-1-63248-169-6-12

created by Van der Poll and Kotze [43]. The Enhanced Established Strategy stems from the Established Strategy for constructing a Z specification and gives guidance and principles to the software engineer when writing a Z specification.

2) Procurement requirements

Partial requirements of our ERP system in this paper are depicted in TABLE II.

TABLE II. PROCUREMENT MODULE REQUIREMENTS LIST

No	Formal Methods
1	The system must be able to keep track of stock for several products
2	Product should have a name, price and quantity of available stock recorded on the system.
3	Each product must have a unique name.
4	User should be able to update products name, price and quantity of stock on hand.
5	The system should have the ability to produce a report with all the products that are below set threshold.
6	The system should allow for the capturing of orders.
7	Once a new order for a specific product is captured it will stay on the "pending" status.
8	All orders on the pending status can be deleted, once deleted the status should change to "Cancelled".
9	A quantity of order should always be more than one.
10	A record of the quantity, price and product name order must be kept.
11	All orders with the status that is pending should be processed when there is required stock on hand.
12	Once the order is processed the status should change to "processed" and the quantity should decrease with same number if products ordered.
13	Customer information needs to be stored and linked to the order. Information includes the name, address and phone number must be stored.
14	One customer can have multiple orders

3) Formal Specification of the Purchasing Module

State Space of Product Schema

Schema *Product* below specifies aspects of products in the ERP system (recall emphasis will be on the procurement part of the ERP).

Product

$products : \mathbb{P}\ PRODUCT$

$produName : PRODUCT \rightarrowtail STRING$

$produPrice : PRODUCT \nrightarrow AMOUNT$

$proQuantity : PRODUCT \nrightarrow \mathbb{N}$

$\text{dom}\ produName = products$

$\text{dom}\ produPrice = products$

$\text{dom}\ proQuantity = products$

Product Schema summary

$\mathbb{P}\ PRODUCT$ represents characteristics of all product detail in the system. Further specification of these characteristics is beyond the scope of this paper.

$prodPrice : PRODUCT \nrightarrow AMOUNT$ is an attribute (monetary amount) of the product and it is declared as a partial function notation ($\nrightarrow$).

$prodName : PRODUCT \rightarrowtail STRING$ denotes the name of the product. Requirement no 3 in TABLE II states that no two products can have the same name, hence a partial injective function is used to specify product names.

The domains are specified in the predicate part of the schema. In this requirement each attribute of the product should equal the identities collection. i.e.

$dom\ produName = products$

$dom\ produPrice = products$

$dom\ proQuantity = products$

State Space of Customer Schema

Before creating an Order schema a customer schema ought to be specified. A customer is linked to an order.

Customer

$customers : \mathbb{P}\ CUSTOMER$

$custoAddress : CUSTOMER \nrightarrow STRING$

$custoPhone : CUSTOMER \nrightarrow STRING$

$\text{dom}\ custoAddress = customers$

$\text{dom}\ custoPhone = customers$

Customer Schema summary

Definition $customers : \mathbb{P}\ CUSTOMER$ represents the set of all existing customers in the system.

Definitions $custoAddress: CUSTOMER \nrightarrow STRING$ and $custoPhone: CUSTOMER \nrightarrow STRING$ are attributes of customers and are specified using a partial function (denoted by $\nrightarrow$). For reasons of space the Order schema which links an order to a customer is not shown in this paper.

The above formal specification fragment shows how the procurement module can be specified using a formal method, in this case Z. Some other formal notations are VDM, B, CSP, OBJ, etc.

Aspects of formal specification illustrated in this section lead to proposition 5:

Prop 5: *Using the Z notation as an entry language ought to facilitate the adoption of formal methods. Z is believed to be easy to learn and apply. Only basic mathematical set theory and logic are required.*

Proc. of the Eighth Intl. Conf. on Advances in Computing, Communication and Information Technology - CCIT 2019

ISBN: 978-1-63248-169-6 DOI : 10.15224/978-1-63248-169-6-12

V. Formal Methods in the commercial world

Since the development of formal methods in the 1980s, their adoption or use within the business arena is slow [10]. Yet, the following software and hardware giants are known to be using Formal Methods:

- Amazon
- Intel
- NASA
- NATS
- Xilinx

Other companies known to also use FMs are: Qualcomm, Nvidia, Cisco, Broadcom, Samsung, Mediatek, AMD, and Huawei. Originally Google's and Microsoft's main foci were software but they starting to develop their own hardware and they have since also adopted formal methods [9]. Start-ups are slowly picking up formal methods as these provide a good return on investment (ROI) with clean code, hence less money is spent on rectifying defects [30].

Next we elaborate on the successful use of FMs by the Intel company

Intel's core business is hardware; for hardware to function correctly, the following needs to be developed: Microcode, Firmware, Protocols and Software. In almost all the products Intel provide, they used to experience problems with the diversity of verification [12]. Consequently, Intel developed various solutions in an attempt to solve verification problems. Their solutions include Propositional Equivalence Inspection (PEI), Symbolic simulation, Symbolic Trajectory Evaluation (STE) and temporal logic model checking.

Intel experienced numerous challenges with some of their products, the most challenging was a physical problem which was the overheating of their Chips and the FDIV bug, which could readily be solved through the use of formal methods. Intel invested over $147 million to cover the cost incurred from chip overheating and the verification problems which also led to the improvements of formal methods within Intel. Intel has realised numerous benefits with using formal methods and they continue to use them on many projects [9].

The above discussions lead to our next proposition:

Prop 6: *Publications of formal methods successes in terms of the money saved in projects, clear specification produced and the overall final product delivered with fewer defects will raise much interest needed for the adoption of FMs.*

Next we combine our propositions and above observations into a formal-methods adoption framework.

VI. Formal Methods Adoption Framework

The proposed framework will be based on the propositions made in this paper and the work that has been done in this knowledge area by other researchers. We first discuss the framework in a tabular format then a graphical presentation of the framework is presented.

A. *Adoption Framework Table*

A framework can be defined as a skeleton or a basic structure of the underlying system [5]. This can be updated as required by adding or deleting items. From a software perspective it can be defined as a set of functions within a system and how they interconnect.

TABLE III presents the proposed framework on the strength of the propositions identified throughout the paper and this assisted in creating our Formal Methods Adoption Framework.

TABLE III. ADOPTION FRAMEWORK

ASPECT	DISCUSSION
Education	Software engineering education in the early stage
	Introduction to formal methods for first year university students
	Universal formal methods standards
	University accreditation specifically on formal methods
	Set theory basics at an early stage of educationssystems
	Step by Step guide on transforming informal requirement to formal specification
	Knowledge sharing and common terminology
Buy-in	Public sector using formal methods for their systems
	Enterprise Top management buy-in
	Project Manager and Senior managers buy-in
	Training companies
	IT community buy-in
	Formal Method language e.g. Z
Remuneration	FM specialist salaries, Scare skill
Environment	IT environments where FMs are going to be utalised
	Tools to write formal specification
	Integration of MS office to formal specification languages
	Open source tools
	Collaborative environment for formal methods specialist to meet
	Built the right attitude within teams, Team buildings
Support Tools	LaTeX, Fuzz,
Publications	Successful use of formal methods should be published regularly
	Forums i.e. internet news letters of formal methods
	Encourage use of formal methods on open source systems
	Library catalogue on formal methods
Results	Positive and negative results should be made available
	Description of each successful component of the system built using formal methods
	System developed using formal methods used in real business environment
	Positive and negative results should be made available

Proc. of the Eighth Intl. Conf. on Advances in Computing, Communication and Information Technology - CCIT 2019

ISBN: 978-1-63248-169-6 DOI : 10.15224/978-1-63248-169-6-12

B. *Adoption Framework Diagram*

Figure 3 is the graphical presentation of the framework in TABLE III. All the steps can be performed in parallel. The larger box represents that the more we focus on that step the higher the probability that the adoption will be a success.

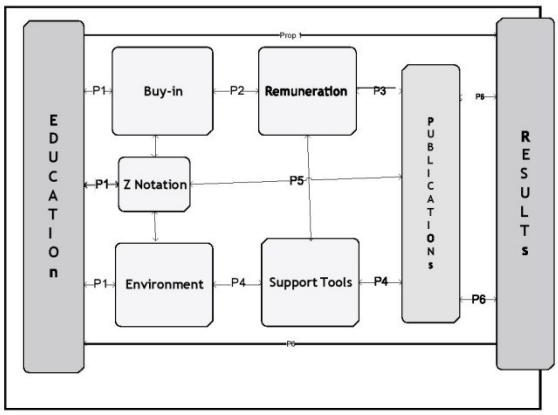


Figure 3. Formal Merhods Adoption Framework

The contents of Figure 3 are discussed next.

Education: This all important aspect forms the foundation to allow for the adoption of FMs in the commercial world. Without dedicated education and the transfer of skills to the new and upcoming engineers the current state of formal methods may well stay as–is. As observed in TABLE III above, education should be from a level as early on as high school. A basic introduction to formal methods should be given, followed at university level by a module on formal methods for first year students. Education will also go a long way to solve the problem of not having a common terminology in formal methods. Students who qualify or pass this course should be recognised by awarding an appropriate accreditation. The Z notation is relatively easy to teach and learn as it requires basic mathematical knowledge. Students and IT professionals might start off by first learning Z and then be introduced to other formal specification languages over time.

Buy-in: IT professionals should be encouraged to use and invest in the use of FMs within their organisations. Buy-in is needed from top management right through to the project manager and the software engineers. Public enterprise buy-in to utilize formal methods in their systems is also important. Furthermore, buy-in from companies that provide IT training courses and the IT community in general will have a major impact on the promotion of formal methods. Established professional societies (e.g. the IEEE) can provide standardized FM teachings in the use and practical application of formal methods. To secure the necessary buy-in from software engineers, training or awareness of formal methods should be provided in a top down approach; i.e. from top management to senior managers, then analysts and then developers. With Z as a recommended formal-methods language for this research, top management need to understand the benefits that Z brings, such as it is easy to comprehend and also the flexibility it inhibits to model a specification that can lead directly to code through successive transformations, and so forth. Many specifiers are familiar with the Z syntax and semantics, hence it's arguably the most used formal language [7].

Remuneration: Good compensation to FM specialists will also attract more people to join the formal methods arena. It should also motivate students and professionals already in the computer science industry to obtain certification in formal methods. Most companies nowadays are concerned with cutting cost and that's what top management understands to place as a priority. Hard evidence of cost saving, reduction in development time and the improved quality of the resultant system when using FMs, should be made available to portfolio- and programme managers.

Environment: Equally important is the environment in which formal methods are to used. The environment should encompasses proper tools to facilitate the use of formal methods. Integration of current software engineering tools with formal methods tools to allow for a smooth transition should be put in place. Teams working on formal methods must have the right attitudes which can also be influenced by the environment they are working in. An encouragement of the use of formal methods in Open source software will bring about new ideas and engender a positive attitude and perception towards formal methods.

Support tools: FMs tools should enhance the UX (i.e. be user friendly) and should allow for an easy integration into current development environment such as the .NET framework and Linux development frameworks. It should also be possible to integrate FMs artefacts and techniques into an existing SDLC. As more and more computer devices are mobile e.g. smart phones and tablets, more Apps for these devices should utilise some form of formalisation. FMs tools should have more automation as well as automated test generation. Lastly, formal methods should facilitate clear estimations on how long it will take to perform analyses. For example, Van der Poll [42] states that "it must be possible to continuously measure the progress archived by formal technique during software development".

Publication: Regular publications on formal methods usage should be encouraged. This can be made available via internet news letters, forums, blogs and a public library (Library catalogue on formal methods). As we can see from the the FMs adoption framework in Figure 3, buy-in and environments feeds into publication(s). A positive buy-in and conducive environment should lead to the successful implementation of formal methods – these should be publicised. In South Africa we have an institute called the CSIR (Council for Scientific and Industrial Research) which is a leading technology research organisation in the country. This body can be used to promote the use of formal methods in software development.

Results: after all the steps have been followed, positive results and findings should be the output. Even negative results should be made known and lessons should be learned through these. Results should lead to the development of the systems using formal methods in the practical world. Each successful component should be described and how success was archived.

Proc. of the Eighth Intl. Conf. on Advances in Computing, Communication and Information Technology - CCIT 2019

ISBN: 978-1-63248-169-6 DOI : 10.15224/978-1-63248-169-6-12

VII. Conclusions and Future work

The use of Formal Methods for software development has been shown to be beneficial, yet their adoption in commercial software development remains slow. Formal methods are still viewed as being difficult owing to their use of mathematical notations. Also, there are many myths that surround the use of formal methods, such as they guarantee a perfect system. Putting more emphasis on education and results will help in the increased uptake of formal methods commercially.

Tools that are readily available and enhance the user experience (UX) will not only encourage specifiers and software engineers to embark on formal methods but aid in producing specifications and systems that are most likely error free. Positive results on the successful utilization of FMs in the commercial system should be made publicly available.

More research and development need to be undertaken to promote the commercial uptake of FMs. Both the public and private sectors should be encouraged to engage with these processes.

A. *Future work*

Formal methods have been around for years and this paper focused on what could be done to increase formal methods usage in commercial world. While this paper made a contribution towards the uptake of FMs through an adoption framework, much work remains to be done.

Common terminology ought to be developed across the academic and industrial formal methods fields. This needs to be widely accepted and standardized across the board. The issue of formal methods tools have been cited by many researchers, tools needs to be developed and these include automated and semi-automated reasoners to discharge proof obligations (POs) resulting from formal specification constructs. Existing tools are either not user friendly or do not perform all the required functionality to write formal specifications [21]. Such tools need to be integrated with current development frameworks such as .NET and JAVA to name a few. Automatic conversion of first-order logic specifications to a full Z specification needs to be looked at. Tools need to be classified and demonstrations of the tools in the form of videos (vodcasts), also indicating the strength of it in practice, ought to be made.

Immediate future work following this paper will be in the form of a survey among academics and practitioners to validate and enhance the framework in Figure 3. Such instrument to assist business in choosing the correct off-the-shelf FMs tools must be made available to business to assist them in their IT operations. More practical examples are needed specifying the advantages and disadvantages of different FMs design methodologies.

Amongst others, the following questions need to addressed:

- How can a formal specification be validated with the user and other stakeholders?
- How can a formal notation be sensibly integrated with more widely used notations such as UML, its class diagrams and use case diagrams?
- How can one automate formal descriptions to generate test cases or even code?

> Without proper education and transfer of skill to the new upcoming software engineers the current state of formal methods will stay as–is.

References

[1] Adesina-Ojo, A. (2011). Towards the formalisation of object-oriented methodologies. University of South Africa, Pretoria. Retrieved from http://hdl.handle.net/10500/11957

[2] Al-Ghofaili, A. A., & Al-Mashari, M. A. (2014). ERP system adoption traditional ERP systems vs. cloud-based ERP systems. In 4th International Conference on Innovative Computing Technology, INTECH 2014 and 3rd International Conference on Future Generation Communication Technologies, FGCT 2014 (pp. 135–139). https://doi.org/10.1109/INTECH.2014.6927770

[3] Atlee, J. M., Beidu, S., Day, N. A., Faghih, F., & Shaker, P. (2013). Recommendations for improving the usability of formal methods for product lines. In 2013 1st FME Workshop on Formal Methods in Software Engineering, FormaliSE 2013 - Proceedings (pp. 43–49). https://doi.org/10.1109/FormaliSE.2013.6612276

[4] Beck, K., Beedle, M., Bennekum, A. Van, Cockburn, A., Cunningham, W., Fowler, M., ... Thomas, D. (2001). Manifesto for Agile Software Development. Retrieved from http://agilemanifesto.org/

[5] Börger Egon, & Stärk Robert. (2003). Abstract State Machines. Springer-Verlag Berlin Heidelberg, 437. https://doi.org/10.1007/978-3-642-18216-7

[6] Bourque, P., & Fairley, R. E. (2014). SWEBOK v.3 - Guide to the Software Engineering - Body of Knowledge. IEEE Computer Society. https://doi.org/10.1234/12345678

[7] Bowen, J. P. (2016). The Z notation: Whence the cause and whither the course? In Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) (Vol. 9506, pp. 103–151). https://doi.org/10.1007/978-3-319-29628-9_3

[8] Bowen, J. P., & Hinchey, M. (2012). Ten commandments of formal methods... Ten years on. In Conquering Complexity (pp. 237–251). https://doi.org/10.1007/978-1-4471-2297-5_11

[9] Cousineau, D., Doligez, D., Lamport, L., Merz, S., Ricketts, D., & Vanzetto, H. (2012). TLA + proofs. In Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) (Vol. 7436 LNCS, pp. 147–154). https://doi.org/10.1007/978-3-642-32759-9_14

[10] Davis, J. A., Clark, M., Cofer, D., Fifarek, A., Hinchman, J., Hoffman, J., ... Wagner, L. (2013). Study on the barriers to the industrial adoption of formal methods. In Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). https://doi.org/10.1007/978-3-642-41010-9_5

[11] Dongmo, C., & Van der Poll, J. A., (2016). Formalising non-functional requirements embedded in user requirements notation (urn) models. University of south africa.

[12] Fix, L. (2008). Fifteen years of formal property verification in intel. In Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) (Vol. 5000 LNCS, pp. 139–144). https://doi.org/10.1007/978-3-540-69850-0_8

[13] Gao, J., Zhang, L., & Wang, Z. (2008). Decision support in procuring requirements for ERP software. In Proceedings of the 9th International Conference for Young Computer Scientists, ICYCS 2008 (pp. 1126–1131). https://doi.org/10.1109/ICYCS.2008.158

[14] George, V. & Vaughn, R. (2003). Application of lightweight formal methods in requirement engineering. Crosstalk: *The Journal of Defence Software Engineering.*

[15] Gibbons, J., & Oliveira, J. N. (2009). Teaching Formal Methods [electronic resource]: Second International Conference, TFM 2009,

ISBN: 978-1-63248-169-6 DOI : 10.15224/978-1-63248-169-6-12

Eindhoven, The Netherlands, November 2-6, 2009. Proceedings / edited by Jeremy Gibbons, JosÃ© Nuno Oliveira. Springer eBooks.

[16] Grabski, S. V., Leech, S. A., & Schmidt, P. J. (2011). A Review of ERP Research: A Future Agenda for Accounting Information Systems. Journal of Information Systems. https://doi.org/10.2308/jis.2011.25.1.37

[17] Hussain, S., Dunne, P., & Rasool, G. (2013). Formal specification of security properties using Z notation. Research Journal of Applied Sciences, Engineering and Technology.

[18] IIBA. (2015). A Guide to the Business Analysis Body of Knowledge (BABOK). International Institute of Business Analysis, Toronto, Ontario, Canada. https://doi.org/10.1017/CBO9781107415324.004

[19] Ilić, D. (2007). Deriving formal specifications from informal requirements. In Proceedings - International Computer Software and Applications Conference. https://doi.org/10.1109/COMPSAC.2007.104

[20] Jaspan, C., Keeling, M., Maccherone, L., Zenarosa, G. L., & Shaw, M. (2009). Software mythbusters explore formal methods. *IEEE Software*. https://doi.org/10.1109/MS.2009.188

[21] Kefalas, P., Eleftherakis, G, A, S. (2003). Developing Tools For Formal Methods. *In Proceedings of the 9th Panhellenic Coneference in Informatics*. http://doi.org/http://dx.doi.org.ezaccess.libraries.psu.edu/10.3149/jms.1403.379

[22] Kilic, H. S., Zaim, S., & Delen, D. (2015). Selecting "the best" ERP system for SMEs using a combination of ANP and PROMETHEE methods. Expert Systems with Applications, 42(5), 2343–2352. https://doi.org/10.1016/j.eswa.2014.10.034

[23] Kneuper, R. (1997). Limits of formal methods. Formal Aspects of Computing, 9(4), 379–394. https://doi.org/10.1007/BF01211297

[24] Li, F.-L., Horkoff, J., Borgida, A., Guizzardi, G., Liu, L., & Mylopoulos, J. (2015). From Stakeholder Requirements to Formal Specifications Through Refinement. In Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) (Vol. 9013, pp. 164–180). https://doi.org/10.1007/978-3-319-16101-3_11

[25] Malik, P., & Utting, M. (2005). CZT: A Framework for Z Tools. In ZB 2005: Formal Specification and Development in Z and B 2005: Formal Specification and Development in Z and B. https://doi.org/10.1007/11415787_5

[26] Markus, M. L., & Tanis, C. (2000). The Enterprise System Experience — From Adoption to Success. Framing the Domains of IT Management: Projecting the Future Through the Past, 173–207. https://doi.org/10.1145/332051.332068

[27] Nwankpa, J. K. (2015). ERP system usage and benefit: A model of antecedents and outcomes. *Computers in Human Behavior*. https://doi.org/10.1016/j.chb.2014.12.019

[28] Palmquist, M. S., Lapham, M. A., Miller, S., Chick, T., & Ozkaya, I. (2013). Parallel Worlds: Agile and Waterfall Differences and Similarities. SEI, Carnegie Mellon University. https://doi.org/CMU/SEI-2013-TN-021

[29] Pressman, R. S. (2009). Software Engineering A Practitioner's Approach 7th Ed - Roger S. Pressman. Software Engineering A Practitioner's Approach 7th Ed - Roger S. Pressman. https://doi.org/10.1017/CBO9781107415324.004

[30] Preuß, S., e, N. A., Gerber, C., & Hanisch, H. M. (2011). Virtual start-up of plants using formal methods. *International Journal of Computer Applications in Technology*. https://doi.org/10.1504/IJCAT.2011.045401

[31] Royce, W. (1970). Managing the Development of Large Software Systems. Proceedings, IEEE WESCON, (August), 1–9.

[32] Seo, G. (2013). Challenges in Implementing Enterprise Resource Planning (ERP) System in Large Organizations: Similarities and Differences Between Corporate and University Environment. *MIT SLOAN School of Management*. https://doi.org/857768973

[33] Shehab, E. M., Sharp, M. W., Supramaniam, L., & Spedding, T. A. (2012). Enterprise resource planning. Business Process Management Journal, 10(4), 359–386.https://doi.org/10.1108/14637150410548056

[34] Sommerville, I. (2016). Software Engineering. Software Engineering. https://doi.org/10.1111/j.1365-2362.2005.01463.x

[35] Spichkova, M. (2012). Human factors of formal methods. In Proceedings of the IADIS International Conference Interfaces and Human Computer Interaction 2012, IHCI 2012, Proceedings of the IADIS International Conference Game and Entertainment Technologies 2012.

[36] Spivey, J. (1992). The Z notation. Prentice Hall International (UK) Ltd. Retrieved from http://www.rose-hulman.edu/class/se/csse373/current/Resources/zrm.pdf

[37] Srihasha, A. V, & Reddy, A. R. M. (2015). Modest Formalization of Software Design Patterns. International Journal of Latest Research in Engineering and Technology.

[38] Steyn, P. S. and Van der Poll, J. A. "Validating Reasoning Heuristics Using Next-Generation Theorem Provers". In The 5th International Workshop on Modelling, Simulation, Verification and Validation of Enterprise Information Systems (MSVVEIS), pp. 43 – 52. Funchal, Madeira, Portugal, June 2007. ISBN 978-972-8865-95-5.

[39] Suryalena. (2013). Enterprise resource planning (erp). Jurnal Aplikasi Bisnis, 3, 145–154.

[40] Toyn, I., & McDermid, J. A. (1995). CADi: An architecture for Z tools and its implementation. Software: Practice and Experience. https://doi.org/10.1002/spe.4380250306

[41] Tretmans, J., & Belinfante, A. (1999). Automatic Testing with Formal Methods. Analysis.

[42] Van der Poll, J. A. (2010). Formal Methods in software Development: A Road Less Travelled. South African Computer Journal.

[43] Van der Poll, J. A., & Kotze', P. (2005). Enhancing the Established Strategy for Constructing a Z Specification. School of Computing,University of South Africa, 0003, (35), 118–131.

[44] Wassyng A., Lawford M. (2003) Lessons Learned from a Successful Implementation of Formal Methods in an Industrial Project. In: Araki K., Gnesi S., Mandrioli D. (eds) FME 2003: Formal Methods. FME 2003. Lecture Notes in Computer Science, vol 2805. Springer, Berlin, Heidelberg

[45] Woodcock, J. I. M., Larsen, P. G., Bicarregui, J., & Fitzgerald, J. (2009). Formal Methods: Practice and Experience. ACM Computing Surveys, 41(4), 40. https://doi.org/10.1145/1592434.1592436

[46] Xilinx. (2005). Xcell Journal, Issue 55 Fourth Quarter 2005. Technology.

Proc. of the Eighth Intl. Conf. on Advances in Computing, Communication and Information Technology - CCIT 2019

ISBN: 978-1-63248-169-6 DOI : 10.15224/978-1-63248-169-6-12

About the Authors:

Aifheli Nemathaga is a qualified software engineer at a South African software company. His research interests are in Formal Methods for correct software development.

John Andrew van der Poll holds a PhD in Computer Science from the University of South Africa (Unisa). He is a full professor at the Graduate School of Business Leadership (SBL) and his research interests are in the construction of highly dependable software for Business ICTs.

Index