

https://dspace.org/

Improvement of the software systems development life cycle of the credit scoring process at a financial institution through the application of systems engineering

Meyer, Nadia

2016-10-11
http://hdl.handle.net/10539/21147

Downloaded from a DSpace repository. https://dspace.org/

Improvement of the Software Systems Development Life Cycle of the Credit Scoring Process at a Financial Institution Through the Application of Systems Engineering

By

Nadia Meyer

A Research Report

Submitted to the Faculty of Engineering and the Built Environment in partial fulfilment of the Requirements for the degree of

Master of Science in Engineering



DECLARATION

Student : **Nadia Meyer**

Student Number : **870445**

Class : **MECN 7018** (Msc. Industrial Eng (50/50))

Cell Number : **082 5551585**

Email : **nadiameyer50@gmail.com**

I declare that this research report is my own unaided work. It is being submitted as a Degree of Master of Science in Engineering to the University of Witwatersrand, Johannesburg. It has not been submitted before any degree or examination to any other university. Due acknowledgement must always be made of the use of any material contained in or derived from this research report.

Signature of Candidate :

Date :

Approvals :

Supervisor	**Name**	**Signature**	**Date**
Advisor/Promoter	Bernadette Sunjka		

ABSTRACT

The research centred on improving the current software systems development life cycle (SDLC) of the credit scoring process at a financial institution based on systems engineering principles. The research sought ways to improve the current software SDLC in terms of cost, schedule and performance. This paper proposes an improved software SDLC that conforms to the principles of systems engineering.

As decisioning has been automated in financial institutions, various processes are developed according to a software SDLC in order to ensure accuracy and validity thereof. This research can be applied to various processes within financial institutions where software development is conducted, verified and tested.

A comparative analysis between the current software SDLC and a recommended SDLC was performed. Areas within the current SDLC that did not comply with systems engineering principles were identified. These inefficiencies were found during unit testing, functional testing and regression testing.

An SDLC is proposed that conforms to systems engineering principles and is expected to reduce the current SDLC schedule by 20 per cent. Proposed changes include the sequence of processes within the SDLC, increasing test coverage by extracting data from the production environment, filtering and sampling data from the production environment, automating functional testing using mathematical algorithms, and creating a test pack for regression testing which adequately covers the software change.

ACKNOWLEDGEMENTS

Throughout my career in finance I have been exposed to systems engineering and systems development life cycles. This has served as a great opportunity to explore the area of systems engineering in greater detail and to gain the necessary knowledge to be able to improve systems and processes. I shall use the holistic approach to fully understand the system and its interactions in future endeavours.

In submitting this work I am greatly indebted to my supervisor Bernadette Sunjka for guiding me closely and encouraging me to follow my own initiatives and ideas. She has been vital in ensuring that I follow the correct approach in my research.

This study would not have been possible without the various experts at the financial institution for participating in the interviews and surveys. Their broad depth of knowledge with regards to each area of the systems development life cycle has enabled me to understand and analyse every aspect of the current system in great detail.

I would like to thank my lecturer Duarte Goncalves for introducing me to hard systems methodologies. I have been able to use his teaching not only in this research but also in various projects at work where systems knowledge is of uttermost importance.

My gratitude to all who participated and contributed in this study will remain a debt I can never repay.

DISCLAIMER AND COPYRIGHT

DISCLAIMER

The acceptance or use of the information provided in this report will in no way relieve the user of its responsibilities in terms of any problem resolution prescription or being accountable for decisions they take in managing their processes. The users shall satisfy themselves that the principles and recommendations used and/or selected are all ways suitable to meet their respective circumstances and context. The onus of ensuring that the principles and recommendations fit the purpose shall at all times rest with the user. Nonetheless, the sole purpose of the information contained in this report is for information and educational purposes.

All statements, comments or opinions expressed in this research report are those of the author and do not necessarily represent the opinions or reflect official position of the University of Witwatersrand or the author's employers.

COPYRIGHT

Except for normal review purposes, no part of this report may be reproduced or utilised in any form or by any means electronic or mechanical, including photocopying, recording or by any information storage or retrieval system without the written consent of the author or the University of Witwatersrand.

TABLE OF CONTENTS

DECLARATION i

ABSTRACT ii

ACKNOWLEDGEMENTS iii

DISCLAIMER AND COPYRIGHT iv

LIST OF ACRONYMS AND DEFINITIONS x

CHAPTER 1 – INTRODUCTION 1

1.1 Research Background and Context 1

1.2 Problem Statement 2

1.3 Critical Research Question 3

1.4 Research Objectives 3

1.5 Outline of the Research Method 4

1.6 Limitations 4

1.7 Outline Structure of Project 5

CHAPTER 2 – LITERATURE REVIEW 6

2.1 Introduction 6

2.2 Software Development 8

2.3 Software Development in Banking 19

2.4 Systems Engineering Development Models 22

2.4.1 The V-model 23

2.4.2 The Waterfall model 25

2.4.3 The Spiral model 27

2.5 Systems Engineering Principles 28

2.6 Framework for Comparative Analysis 35

2.7 Summary of the Literature Review 58

CHAPTER 3 – RESEARCH METHOD 59

3.1 Introduction 59

3.2 Research Design 60

3.3 Purpose of the Comparative Analysis 60

3.4 Research Instrument and Methodology 61

3.4.1 The interview process 61
3.5 Data Analysis 72
3.6 Reliability and Validity 73
3.7 Limitations 76
3.8 Ethical Considerations 76
3.9 Chapter Summary 77
CHAPTER 4 – DATA ANALYSIS AND RESULTS 78
4.1 Introduction 78
4.2 Participant and Implementation Overview 79
4.3 The Current Software SDLC 80
4.3.1 The credit scoring process for the case site 81
4.3.2 The software development model – current state analysis 83
4.3.3 The software systems development life cycle – current state analysis 93
4.4 Comparative Analysis 107
4.5 Chapter Summary 130
CHAPTER 5 - DISCUSSION 131
5.1 Introduction 131
5.2 Comparative Analysis Discussion 131
5.2.1 Definition and analysis of requirements 131
5.2.2 Optimal design of the system 132
5.2.3 Optimal integration and development of the system 133
5.2.4 Optimal verification and validation of the system 134
5.2.5 Successful implementation of the system 135
5.2.6 Intensive risk management during the entire SDLC 136
5.3 Comparative Analysis Implications 137
5.4 Chapter Summary 141
CHAPTER 6 – CONCLUSION AND RECOMMENDATIONS 142
6.1 Conclusions 142
6.2 Implications of the Research 143
6.3 Limitations 143
6.4 Recommendations for Future Work 143
BIBLIOGRAPHY 144

APPENDICES 150
APPENDIX A – *Interview Introductory letter* 151
APPENDIX B – *Interview Questions* 153
APPENDIX C – *Interview Transcripts* 154

LIST OF TABLES

Table 4.1 Participant information 80
Table 4.2 Comparative Analysis Results 108
Table 5.1 Recommendations 138

LIST OF FIGURES

Figure 1.1 Study Outline 5
Figure 2.1 Literature Review Roadmap 7
Figure 2.2 The V-model (Kasser, 2007) 23
Figure 2.4 The Spiral Model (Boehm, 2000) 27
Figure 3.1 The Interview Approach (Wilkinson & Birmingham, 2003) 63
Figure 3.2 Determining Interview Themes for the Study 64
Figure 4.1 Results Road Map 79
Figure 4.2 The Credit Scoring Process 81
Figure 4.3 The V-model (Mathur & Malik, 2010) 83
Figure 4.4 The Software SDLC Phases 94
Figure 4.5 The Software SDLC Phases in Detail 94
Figure 4.6 Overview of the Software Development Procedure 95
Figure 4.7 The Requirements Analysis Phase 95
Figure 4.8 The Analysis and Design Phase 99
Figure 4.9 The Development Phase 101
Figure 4.10 The Testing Phase 103
Figure 4.11 The Implementation Phase 107

LIST OF ACRONYMS AND DEFINITIONS

Customer management strategy (CMS): The core scoring area across the bank and interfaces with many other systems. CMS has different systems enabling the automated credit application process, automation and control of the daily cheque account excesses, automatic renewal and referral of overdraft limits and generation of consumer triggers for improved efficiency and risk control together with the functionality of behavioural risk scoring. The systems are Online(Real-time) and batch systems. CMS is responsible for the scoring across the bank for all credit applications. It has a centralised storing collection of all credit applications. It facilitates the use of the scoring rules and scorecards (scoring logic).

Hogan® Systems Core Technology Services: An information technology (IT) service provider within the bank that supplies IT services and solutions to various business units. It runs, monitors and supports systems.

Hogan Technology Quality Assurance (HTQA): Department responsible for providing testing services to Hogan. It supports business units and channels interfacing with Hogan from a testing and quality assurance perspective. It is responsible for the end-to-end testing of new and existing functionalities to ensure that they are in accordance with the business requirements and specifications.

Experian Strategy Management (SM): Software warehousing the logic for all credit scoring decisions in the production (live) environment. This scoring engine uses information relevant to the application and facilitates the rules and scorecards to receive a result/decision. These rules are owned by Dynamic Decisioning. A Rules Based Decision document is housed by Dynamic Decisioning that contains the business logic within the Strategy Management Generation 3 (SMG3) system for the scoring decisions.

Dynamic Decisioning (D2): Department responsible for the software coding of the credit scoring logic, which includes the scoring rules and scorecards, into SM.

Integrated deposit system (IDS): Information system that contains any information on a cheque/overdraft account with reference to account information versus personal information which is kept on customer information systems (CIS). Therefore, it contains information such as transaction history, statistics, overdraft information, fees and charges, accrual buckets, etc.

Customer information system (CIS): Information system contains customer demographic/profile information.

Integrated loan product (ILP): Information system that contains a month-end snapshot of sole owned integrated loans processing account details. These loans include personal loans (PLS) and mortgage loans (MLS) for a bank's companies/subsidiaries.

Investment product house (IPH): Information system that contains information on all savings and investment accounts for a bank's companies/subsidiaries.

Online: Real time processing of credit applications.

Scoring: Consumers are assessed for affordability and risk for their credit applications.

Result: Final output data from SMG3.

Decision: An approved, referred, or declined decision passed from SMG3.

Pre-bureau: SMG3 contains rules to determine whether a customer is eligible for the loan before costs are incurred to send the send the customer data to the credit bureau.

Post-bureau: SMG3 contains rules based on credit bureau information to determine whether a customer is eligible for a loan.

Payment profiles: A customer's financial history and repayment information on all credit loans.

Single application database (SAD): Single application database that stores all customer credit application information. A SAD record is created once a customer applies for a credit loan.

Central bureau mainframe (CBM): Database that stores credit bureau information received from the credit bureau.

Mainframe bureau gateway (MBG): Database that stores credit bureau information received from the credit bureau as well as the aggregations of the information/data.

Aggregations: Roll up of a customer's explicit credit bureau data. For example, worst arrears in the last six months.

Scorecard: A mathematical model that has a combination of variables that predict the probability a customer has of defaulting on a loan. This probability of default is mapped to a score.

CHAPTER 1 – INTRODUCTION

1.1 Research Background and Context

The financial industry has evolved remarkably with the advancements of technology in recent years. A decision on a credit application was previously a manual process where a staff member assessed each application and made a decision based on his or her own judgment. With the rise of technology, credit applications are assessed by logic contained in software programs known as a credit scoring process.

Software development and the management thereof form an integral part not only in credit scoring but also across all sectors in financial institutions today. Ismail (2012) explains that in financial institutions all means of business occur through electronic transactions. He therefore emphasises the importance of the integrity of software systems as improper management of the software systems could impact daily operations significantly.

Tanrikulu and Ozcer (2011) confirm this view and emphasise the importance of systems management by explaining that as today's banking industry relies heavily on information systems for most of its functions, the organisational complexity involved in system development requires well-established processes and the proper execution thereof. They suggest that a high percentage of software system projects fail due to poorly defined processes that reside in the system development process. Rajkumar and Alagarsamy (2013) estimate that between 50 and 80 per cent of software development projects fail where the software deliverable did not meet the user requirements. They attribute this to several factors including lack of customer involvement, unclear objectives, poor requirement set, lack of resources, failure to communicate and act as a team, project planning and scheduling, cost estimation, inappropriate estimation methodology, cost estimation tools, poor testing, risk management and unrealistic expectations. Kaur and Sengupta (2011) summarise these opinions by stating that an overall poor project management process is to blame for project failure.

The case site is the central management unit in the banking institution responsible for the development and maintenance of the software scoring system, which is known as the credit scoring process. When a customer applies for a credit loan at the banking institution, the his or

her credit feasibility is determined by this process. This system includes the logic for scoring such as the credit scorecard, limit and affordability calculations, and scoring rules. It often occurs that changes are required to the current credit scoring logic in order to improve scoring accuracy.

The software development process that takes place in order to make the requested change is called the systems development life cycle (SDLC), which has been developed on the principles of systems engineering (SE). The SDLC is a well-known software development method. McMurtrey (2013) asserts that the SDLC to is a well-tested methodology for software development. Areas of concern in the current SDLC include integrity of verification, life cycle schedule and cost.

As the integrity and schedule of the credit scoring software is of utmost importance in order to ensure decision accuracy, the establishment and execution of the proper processes are required. This study therefore examines the systems engineering principles pertaining to SDLCs in order to seek ways to improve the current format.

1.2 Problem Statement

It often occurs that changes need to be made to the current credit scoring logic. In order to make the necessary changes, a project is logged and initiated so that new software can be developed with the required changes.

The current software SDLC takes approximately 90 days to complete. Management wishes to investigate whether the life cycle can be reduced without compromising on quality. The quality of verification of the system is of concern as functional testing and regression testing is highly reliant on unit testing being performed correctly. Functional testing and regression testing are not performed on a sufficient number of customers that is representative of the customer database. The test cases also do not cover all scenarios of the software change. Thus functional as well as regression testing is not performed effectively.

Cost is of concern in the current SDLC as six testers are required in order to perform functional testing. Therefore the capacity utilisation is greater than 300 per cent. Management is therefore

required to hire or assign more testers in order to increase the coverage of tests in the allocated time.

The purpose of the study is to analyse the current software SDLC, compare the SDLC to systems engineering principles, and improve the SDLC by addressing the inefficiencies found. The current software SDLC requires analysis and investigation in order to see whether the system can be improved in terms of cost, schedule, and quality.

1.3 Critical Research Question

How can the software SDLC be improved in terms of cost, schedule, and quality for a credit scoring system using systems engineering principles?

1.4 Research Objectives

The research objectives were to:

- Identify areas within the current SDLC used by the case site that can be improved in terms of cost, schedule, and performance by performing a comparative analysis between the current software system development process and a recommended SDLC that complies with SE principles.
- Recommend changes, based on SE principles, to the current software system development process that will assist in the improvement in terms of cost, schedule, and performance.

1.5 Outline of the Research Method

The research method consisted of the following consecutive phases:

1. Gaining a detailed understanding of the current SDLC by conducting face-to-face semi-structured interviews with experts
2. Documenting the current software SDLC from information obtained during interviews
3. Mapping the current software SDLC process through visual sense-making from information obtained during interviews
4. In-depth literature analysis at academic level
5. Comparative analysis between current software SDLC and literature review
6. Identification of inefficiencies in the current software SDLC according to the literature reviewed
7. Recommendations on improving the inefficiencies in the current software SDLC based on information obtained in the literature review and brainstorming techniques

1.6 Limitations

Current data security and privacy protocols as per the bank's standards would apply with the proposed changes. The disclosure and privacy of data would also have to adhere to the Consumer Protection Act 68 of 2008 (CPA) and Payment Card Industry (PCI) regulations. The proposed SDLC would be dependent on the interface areas being able to provide the resources necessary for the development and implementation of the changes identified for those interfaces. It would also be dependent on the interface areas being able to deliver on time.

The proposed SDLC would contain several risk factors. Performance issues could pose a problem due to the increase in volume of data. There could possibly be a lack of capacity as several additional interfaces and processes would be required. Data privacy and security could be compromised should the method of depersonalising the data not be effective.

The research has been limited to a single case site. The case site selected is a central management unit within the financial institution, which is responsible for the entire development and maintenance of the credit scoring process.

1.7 Outline Structure of Project

Figure 1.1 summarises the key chapters of the study.

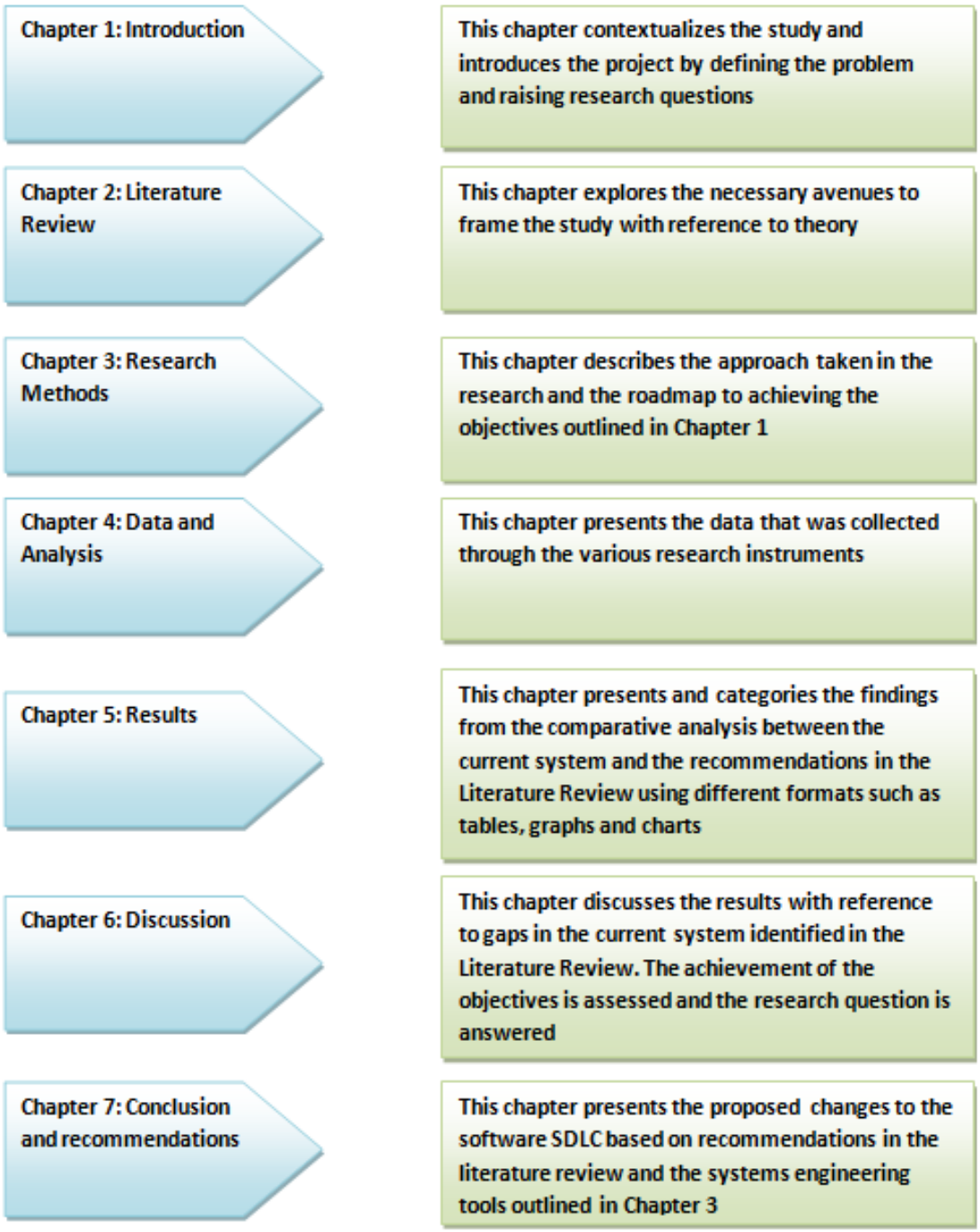


Figure 1.1 Study Outline

CHAPTER 2 – LITERATURE REVIEW

"Research is to see what everybody else has seen, and to think what nobody else has thought."

~ Albert Szent-Gyorgyi

2.1 Introduction

The purpose of this literature review is to investigate the SE principles applied to SDLC in order to perform a comparative analysis between the current SDLC and systems engineering principles. Areas of inefficiencies in the current SDLC will be identified and the recommendations from the literature review will be used to improve the current SDLC in terms of cost, schedule and quality.

The chapter is divided into six parts:

- A high level analysis of software development
- A high level analysis of software development in banking
- A review of the general development models used in software development projects
- An overview of Systems Engineering as a subject
- A high level analysis of systems engineering applied to systems development life cycles
- The construction of a framework for the comparative analysis using systems engineering principles

The emphasis of the literature review is on the application of SE on SDLC, but it will also deal with successes and general problems experienced therein. The goal is twofold:

1. Identification of successes and issues experienced in software development life cycles
2. Integration of the different principles and methodologies emanating from systems engineering applied to software development life cycles and incorporating other methods and recommendations.

Figure 1.2 depicts the outline of the literature review presented in this chapter.

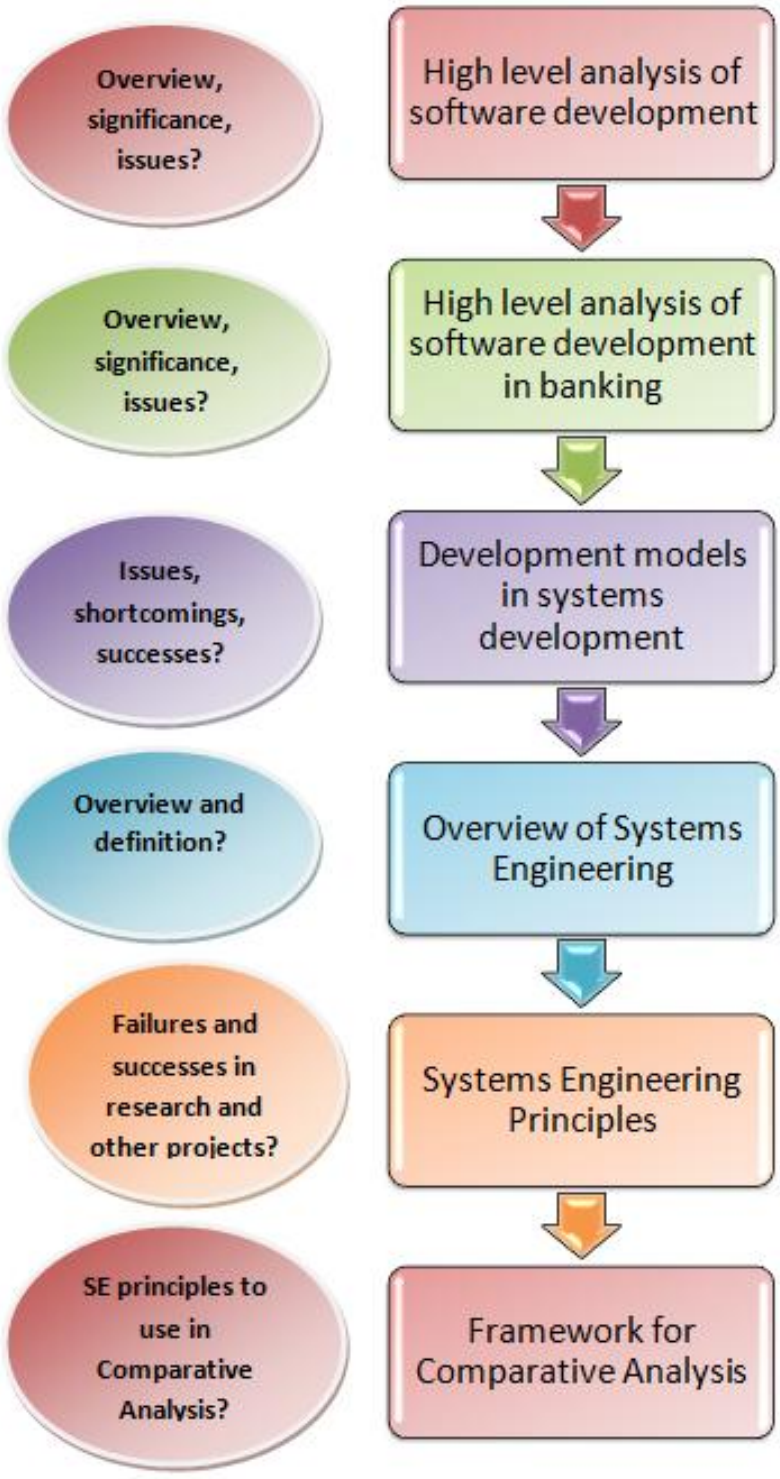


Figure 2.1 Literature Review Roadmap

2.2 Software Development

Maier (1999:268) describes a system as “an assemblage of components that produces behavior or function not available from any component individually”. The author concludes that a system is therefore independent and operates and functions independently with its own purpose. He explains that it can be integrated into a larger system where its purpose, operations, processes and functionality might change to fit in with the larger system to meet the objective of the larger system. Buede (2009:3) defines a system as “a collection of hardware, software, people, facilities, and procedures organized to accomplish some common objectives”. He suggests that the objectives of a system are usually concerned with cost, schedule and performance.

Software has become a central and critical factor in many fields in order for companies and institutions to remain competitive and to improve quality of life. Sweis (2015:174) defines a software system as “software that stores, retrieves and disseminates information, thereby supporting people and organizations and helping to accomplish their work efficiently”.

With the increase in software users across multiple industries worldwide and the increase in complexity of these software systems, SE development has become more complex. Therefore, it requires a more advanced development approach and the integration of different areas (Okafor, 2011). The author explains that this has resulted in problems such as delays in the completion of projects within deadlines, failure to adhere to budgets and less than satisfactory quality.

Iyakutti and Alagarsamy (2011) define a project as work that is managed by a life cycle and concludes when specific objectives have been attained. Sweis (2015) views project success in terms of three aspects that are time, cost and quality. The author views project success as delivering the project within the required time, cost and quality according to the users’ needs. This aligns with Hijazi et al. (2014) who state that cost and time are the main causes for project failure.

Patil and Yogi (2011) agree with this view and state that in software development project failure due to untimely completion of projects or overspending of costs remains a common occurrence. The authors explain that project failure is often associated with the inaccurate estimation of effort and resources due to factors such as inexperience, unclear requirements, unfamiliar future technologies, development environment and complexities in design and development.

They explain that there are four variables that typically control software projects: time, requirements, resources and risks. Boehm (2000) defines risk as the probability of an event occurring that will cause the project to fail.

Patil and Yogi (2011) conclude that it is therefore crucial to make accurate estimations regarding time and resources as any unexpected changes to the four variables could result in project failure. In order to make accurate estimations, risk management is crucial. “Therefore, every project manager is required to perform methodical investigation of risk in order to avoid cost and schedule overruns” (Patil & Yogi, 2011:262). Through their research, the authors found the most critical risk factors to be lack of top management commitment to the project, failure to gain user commitment, and misunderstanding the requirements.

In order to understand software development and the risks involved, an understanding of the SDLC is required. According to Hijazi et al. (2014):

> Software development process or software development life cycle (SDLC) is a structure imposed on the development of a software system, according to this structure the software development process involves five different phases: Requirements Analysis and Definition, Design, Implementation and Unit Testing, Integration and System Testing, and the Operation and Maintenance phase.

Ragunath et al. (2010) describe an SDLC as a software cycle consisting of various parts and phases that describe or prescribe how software should be developed. Hijazi et al. (2014) state that various risks are involved throughout the SDLC and adequate understanding of the problems and risks are required. They list the risks that are common to all SDLC phases as: continually changing requirements, compromising on quality due to time contention, project funding loss, data loss due to poor documentation and team turnover and miscommunication between people.

Khan et al. (2014) suggest that uncertainty within a project poses a major risk for quality project completion within the allocated time and budget. They further suggest that uncertainty can be related to organisational politics, stability of the organisational environment and the organisational support for a project.

Patil and Yogi (2011) contend that project managers should spend sufficient time understanding the project complexities, choosing resources, discussing the project with experts and learning from past experiences. Critical risk factors that could influence the project should be identified

and listed. Risks should then be analysed, prioritised and subsequently mitigated and resolved where necessary (Patil & Yogi, 2011).

Swarnalatha et al. (2014) emphasise the importance of proper requirements gathering and analysis. They state that well defined requirements are critical in meeting clients' needs. The authors define the process of requirements gathering and analysis as requirements engineering: "Requirement engineering is a practical and systematic approach through which the software or system engineer collects functional or non-functional requirements from different customers and design and develop them into the quality software development processes" (Swarnalatha et al., 2014:5045). In their framework for software requirements engineering they propose the following steps in order to avoid defining improper requirements:

- Identify the clients of the system and collect the raw requirements (functional and non-functional) from all points of view through observing and interviewing
- Develop standards and constraints in order to ensure common understanding
- Analyse requirements by comparing it to user or business requirements or objectives
- Prioritise requirements
- Document requirements for future reference
- Organise and define requirements according to a hierarchy from high-level requirements that address business objectives to low level requirements that address component objectives
- Perform dynamic allocation by assigning the functional and non-functional requirements to the relevant system elements
- Validate and verify requirements by reviewing them with clients, prototyping according to requirements and comparing system documentation to clients' objectives.
- Perform software requirements management by keeping track of and documenting all interrelationships and dependencies of software requirements changes.

Sommerville (2006) cited in Hijazi et al. (2014) recommends conducting a feasibility study during the requirements analysis phase in order to determine whether the software system is possible and necessary to construct as well as to identify possible risk factors in the development and deployment of the system. The author lists common risk factors related to a feasibility study:

- Inadequate estimation of project time, cost, scope and other resources

- Unrealistic schedule causing project managers to overload resources in order to meet project deadlines
- Unrealistic budget
- Unclear project scope as a result of project managers not having a clear and detailed understanding of the project
- Insufficient resources

Khan et al. (2014) share similar thoughts to those listed above. They state that proper planning and control of a project are critical risks to project success. They state that poor planning and control lead to poor resource planning and allocation which in turn lead to unrealistic schedules and budgets. In his research, Sweis (2015) found that the underestimation of timelines, poor internal communication, incorrect assumptions regarding resource availability and weak definitions of requirements and scope were some of the main factors responsible for project failure.

Sommerville (2006) cited in Hijazi et al. (2014) further recommends that requirements elicitation should be performed where the system deliverables, performance and constraint requirements are gathered, reviewed and articulated with the help of the different stakeholders of the system. Hijazi et al. (2014) list the common risk factors related to requirements elicitation:

- Unclear requirements that are not understood by the analysts and developer
- Incomplete requirements that are missing some of the user needs, constraints and other requirements
- Inaccurate requirements that do not adequately reflect the users' needs
- Ignoring non-functional requirements by placing more emphasis on the functional aspects of the system
- Conflicting user requirements
- Unclear description of the real environment wherein the software system will operate
- Gold plating by adding additional functionality to the system in order to improve the system that was not part of the initial scope

Sweis (2015) asserts that incomplete specifications at the outset of a project is one of the main causes for project failure.

Sommerville (2006) cited in Hijazi et al. (2014) recommends conducting a requirements analysis by "analyzing, classifying, organizing, prioritizing, and negotiating the stated requirements"

(Hijazi et al., 2014:217). Hijazi et al. (2014) list the common risk factors associated with requirements analysis:

- Non-verifiable requirements where ‘a finite cost effective process’ can’t be used to verify or validate the requirements
- Infeasible requirements where the project can’t be implemented within the constraints of the project. For example, lack of resource availability could compromise the project implementation
- Inconsistent requirements that contradict with other requirements
- Non-traceable requirements where the source of origin is unknown
- Unrealistic requirements that are not ‘clear, verifiable, accurate, consistent, complete and feasible’

Khan et al. (2014) state that uncertainty in requirements pose a major risk to project success. They contend that the frequent changing of requirements and inadequate, unclear, ambiguous and unusable requirements are common risk factors related to requirements gathering and analysis.

Sommerville (2006) cited in Hijazi et al. (2014) recommends validating requirements by ensuring that the stated requirements define what the users want. Li (1990) suggests that the requirements should be verified by manually comparing them to the service request and the current users’ opinions. He further suggests that if any discrepancies are found, the requirements definition and analysis procedure should be redone. Hijazi et al. (2014) assert that common risk factors associated with the validation of requirements include terminology developed by technical resources that are misunderstood by the end users, and expressing user requirements in natural non-formal language which cause misinterpretation between users and other resources.

Lastly, Sommerville (2006) cited in Hijazi et al. (2014) recommends documenting requirements in order to serve as a means of communication between stakeholders. Hijazi et al. (2014) state that the common risk factors associated with the documentation of requirements are inconsistent documentation that does not correlate with stated requirements, and creating non-modifiable requirement documents that are difficult to maintain.

During the design phase, the system architecture is established. Sommerville (2006) cited in Hijazi et al. (2014) states that the design phase involves “examining the requirements document

(RD), choosing the architectural design method, choosing the programming language, constructing the physical model, verifying, specifying, and documenting design activities" (Hijazi et al., 2014:219).

Hijazi et al. (2014) recommend that the examination of the RD should be carried out by the developer in order to ensure the person developing the system understands the requirements. The authors state that a common risk factor associated with examining the RD is that requirements are not clear for the developer due to lack of involvement during the requirements analysis and definition phase.

Choosing the architectural design method involves defining the software components in a systematic way based upon the project's needs. "Choosing the programming language should be made early in the design phase as soon as the architectural design method is chosen, since it should support it" (Hijazi et al., 2014:220). The authors recommend choosing the programming language carefully according to the needs of the project and the architectural design method.

Hijazi et al. (2014) list risk factors associated with the construction of the physical model:

- A large and complex software system which could prove difficult for developers to decompose
- A complicated design, which is not understandable
- Large size components that result in difficulty in determining the functionality of the component and assigning functions to the components
- Unavailable expertise for reusability that result in risk due to resources involved in the previous design no longer being available
- Less reusable components than expected as a result of inaccurate estimation of available reusable components during the requirements analysis phase. This could result in increase development time as components will need to be built from scratch

Khan et al. (2014) mention project complexity as a major risk to project success. They state that project complexity is due to factors such as large number of interfaces between components, the use of new technologies and complex automated processes.

Hijazi et al. (2014) state that the design of the system should be verified in order to ensure that it meets the stakeholders' requirements. The authors mention risk factors that are associated with verification include difficulties in verifying design to requirements by the developer, too many

feasible solutions to the same design problem (which could make it difficult to choose the correct one), and incorrect design that does not match some or all of the requirements.

Hijazi et al. (2014) list risk factors related to the specification of the design activity:

- Difficulties in allocating functions to components due to incorrect system decomposition and ill-defined components and requirements
- Unnecessary extensive specification of modules processing which result an unnecessary large design document
- Omitting data processing functions by poorly defined functional definitions
- Poor management of large amounts of passing data to be used by other components in the component hierarchy leading to poor readability and confusion

Hijazi et al. (2014) mention the risk factors to be avoided during the documentation of the design:

- Incomplete design document that lacks detail necessary for programmers to work independently
- Large design document that include extensive unimportant information
- Unclear design document where the components are poorly defined and the document is written in uncommon natural language
- Inconsistent design document as a result of duplication and overlapping between components

Hijazi et al. (2014:223) describe the implementation and unit testing phase as “where the programming takes place in order to execute the previously defined design as a set of programs or program units”. They define the programming or coding of the software as “the process of writing design modules in the predefined programming language; this includes developing the user interfaces”. They outline several risk factors related to the programming (coding) of the software:

- Non-readable design document that is too large or unclear making it difficult for the programmers to understand
- Programmers cannot work independently due to incomplete design document causing the programmers to make their own decisions regarding certain components
- Developing the wrong user functions and properties due to non-readable, inconsistent or incomplete design document

- Developing the wrong user interface due to poor understanding of the users' needs and a poorly detailed design specification
- Programming language does not support the architectural design due to the language not being chosen early in the design phase according to the architecture
- Modules are developed by different programmers resulting in inconsistent, complex and ambiguous code
- Complex, ambiguous, and inconsistent code due to programmers not following coding standards and best practices in programming
- Different versions for the same component developed by different programmers in the team causing integration problems
- Developing components from scratch resulting in increased time and effort
- Large amount of repetitive code that results in increased time, effort and cost
- Inexperienced programmers resulting in complex and ambiguous code as well as wrong functions, properties and user interfaces
- Too many syntax errors due to the programming language being sensitive and having poor quality compilers and debuggers
- New and unfamiliar technology used in the project causing developers to experience difficulties in programming accurately and efficiently

Sommerville (2006) cited in Hijazi et al. (2014) describe unit testing the process in which each source code module is tested separately in order to verify that it is performing according to the specifications before integrating the different components. Li (1990) suggests that unit or module testing should be performed during the development phase and by the resource that coded, and therefore understands, the internal details of the module. Yoon (2013) agrees with this view and suggests that test cases should be designed by the programmer and should be automated to allow for ease of reuse. Hijazi et al. (2014) mention several risk factors related to unit testing:

- High fault rate in newly designed components
- Code is not understandable by reviewers resulting in developers struggling to fix errors that caused the component defects
- Lack of complete automated testing tools resulting in a boring and monotonous testing process and poor results
- Informal and ill-understood testing process resulting in intuitive techniques being used, which could result in poor verification and validation of the components

- Not all faults are discovered in unit testing due to lack of testing automation and inappropriate testing techniques
- Poor documentation of test cases resulting in lost knowledge for future use
- Data required by a module under test from other modules through lack of use of coding drivers and stubs
- Coding drivers and stubs resulting in additional defects, time and cost
- Poor regression testing due to solely selecting the original test cases

During integration phase the unit software modules are iteratively integrated and tested to produce the complete software system (Hijazi, et al., 2014). Li (1990:27) describes integration testing as "a process of merging and testing program modules to see if they can work correctly as a whole without contradicting the system's internal and external specifications". He suggests that integration and integration testing should be performed according to the structure of the system. He further suggests that the test cases and data should be based upon the system specifications. The following risk factors during this phase are identified by Hijazi et al. (2014):

- Difficulties in ordering components' integration due to not performing integration incrementally or integrating in the wrong order
- Integrating the wrong versions of components or the wrong components resulting in compromised functionality
- Omissions of important components resulting in compromised functionality
- Data loss across component interfaces due to the mismatch between the number and order of parameters between components
- Difficulties in localising errors when integration is not performed incrementally
- Difficulties in repairing errors once the system has already been integrated as it is difficult to detect where the error occurred as well as difficult to prevent secondary errors resulting from a fixed defect

Hijazi et al. (2014:227) describe systems testing as: "The integrated software is tested to ensure that the software system meets the software requirements and system". Li (1990:27) describes system testing as "verifying that the system as a whole is structurally and functionally sound". He states that the data and test cases for structural and functional testing should be designed according to the system requirements specifications. He recommends using "real-life data as the test data and the test results of the new system can be easily verified by the actual results of the old system".

Hijazi et al. (2014) mention several risk factors related to system testing:

- Unqualified testing team and programmers who misuse the available tools, resources and techniques
- Limited testing resources such as time, budget and tools
- Inability to test the operational environment due to difficulties in delivery and installation within time and budget constraints
- Incomplete testing by testers due to too many possible variables, combinations, sequences, configurations and interactions
- Testers relying on process myths and designing their test cases on old requirements that were established early in the SDLC and not the latest requirements that are representative of the users' needs
- Wasting time in building testing tools rather than doing testing
- The system being tested is not testable enough due to unverified requirements and poor application of quality assurance principles

Parvez (2012:339) describes regression testing as:

> …re-testing an application after its code has been modified to verify that it still functions correctly. Regression testing consists of re-running existing test cases and checking that code changes did not break any previously working functions, inadvertently introduce errors or cause earlier fixed issues to reappear.

The author suggests that the test pack should be continuously updated with the latest relevant test cases and that the execution of testing should be automated.

The importance of the team members in project success should also be emphasised (Khan et al., 2014). Risk factors include turnover, insufficient knowledge, cooperation, communication and motivation among team members.

Hijazi et al. (2014) describe the operation and maintenance phase as implementing the complete and integrated system into a production environment, where it is tested and maintained. They mention several risks related to this phase:

- Problems in installation due to inexperience, inadequate knowledge of the system's nature and function, and a complex system deployed in a challenging environment
- An effect on the production environment as the system is deployed

- A change in the environment resulting in the change of software behaviour causing the system to not be able to be deployed correctly
- New requirements emerging while operating the system in order to meet the current actual user needs, business, environmental, and organisational changes
- End users experiencing prolonged difficulties in using the system that thus threaten the acceptability of the system
- User resistance to change due to being excluded from the process of making changes made regarding the performance and behaviour of the system
- Missing capabilities that the users expect to find
- Too many software faults that were not discovered and corrected earlier in development
- Testers not performing well due to problems in the operational environment, unqualified management, lack of tools and testing resources, and lack of the involvement of different system stakeholders
- Difficulty in deciding whether to suspend or resume acceptance testing with the discovery of defects
- Insufficient data handling by the software system due to large amounts of data in the production environment that cannot be handled by the system
- The software engineer cannot reproduce the problem experienced by the end users and can therefore not find the exact cause of the problem. This could be due to users not describing the problem in sufficient detail
- Problems in maintainability due to system constraints and rigid architecture
- Budget contention where the budget did not cater for repeating the activities in software development during implementation and maintenance. This could result in important activities such as verification and validation of the system to be excluded.

Khan and Khan (2014:121) describe acceptance testing as: “Formal testing with respect to user needs, requirements, and business processes conducted to determine the acceptability of the system”. The acceptance of the system by the end users is of critical importance for project success. Li (1990:27) states that the purpose of acceptance testing “is to ensure that the software system meets the previously defined system external specifications, acceptance criteria and system requirements definition before it is installed, integrated and checked out in the operational environment”. He suggests that acceptance testing should be performed by the end users with the help of the development group and the test cases and data for implementation and acceptance testing should be developed based on the system

specifications and the user criteria. In order to increase the probability of acceptance, he suggests that the end users should be involved in the early phases of the life cycle in order to accept the implemented system.

Once the system has been accepted, the end users should provide formal acknowledgment thereof (Li, 1990). This recommendation is supported by Khan et al. (2014). The authors state that the lack of user involvement throughout the development of the software poses a major risk to the project success. Sweis (2015) mentions additional factors related to user acceptance that are responsible for project failure such as a high degree of user customisation during application, changes in design specification late in the project, lack of user involvement and inputs from the onset and changes in key individuals such as the project manager.

From looking at the risk factors, it is clear that the verification and validation of activities within the SDLC are critical in order to prevent defects at all stages of the SDLC. It can therefore be concluded that testing forms an integral part of software development. Li (1990) describes testing as the procedure to find errors and to see if the software is and is not doing what it is intended to do. He suggests that testing should start immediately once the project is initiated. Tuteja and Dubey (2012) agree with this notion and recommend that software testing should be performed as early as possible in the SDLC, preferably during the requirements analysis phase, and should be performed during each phase and activity of development. Khan and Khan (2014) state the benefits of testing as early as possible within the SDLC include resolving small problems before they become bigger problems, obtaining and understanding important quality attributes and resolving issues.

2.3 Software Development in Banking

In their research, Tanrikulu and Ozcer (2011) found that the banking and financial services industry relies heavily on software systems for most of its functions and has the highest software expenditure amongst industries. Iyakutti and Alagarsamy (2011) state that all industries, including banking, have certain SDLC standards in common. In their research, Chomal and Saini (2014) found the major causes of software project failure across all industries to be:

- Large software projects with a large degree of complexity
- Lack of user or customer involvement and contribution throughout the project
- Estimating the project schedule based on time required to complete tasks instead of the actual time spent on tasks
- Inadequate monitoring of and communication in projects resulting in poor requirements understanding and project evaluation
- Poor requirements gathering and definition
- Lack of testing resources and poor knowledge and skills of resources
- Failure to meet with staff in order to establish proper requirements
- Failure in considering resource capabilities during project schedule planning resulting in inaccurate estimation of project schedule
- Failure in considering budget and costs during project planning that result in incompletion of the project or excluding important tasks
- Poor project management such as lack of creating work breakdown, lack of clearly defining and assigning duties and responsibilities to resources over time and lack of proper time and resource scheduling.
- Lack of risk management to identify concerns before they become problems that affect the project delivery
- Unrealistic project expectations
- Not identifying and analysing all users and their needs
- Lack of using good engineering standards during development
- Lack of assessing resource knowledge and capabilities related to new technology used in the project which leads to inaccurate time estimations
- Lack of top management commitment and loyalty to the project

Iyakutti and Alagarsamy (2011) recommend that the requirements phase in banking should include analysis of the client's requirements, assessment of the current environment, quantifying performance and interface requirements, and developing business requirements that will satisfy the needs of the business. They recommended that assessments and reviews be conducted in order to identify possible risks early in the life cycle. They further recommend validating and verifying the phase by documenting the requirements definition, analysing the cost and benefits, reviewing the preliminary project plan, and reviewing risks and contingency plans. Through their research, Tanrikulu and Ozcer (2011) found that a software development problem common to banks related to the requirements analysis phase is that a formal process is not used to track

and control changes in the systems requirements specification (SRS) documents. This could lead to design and development according to outdated specifications.

Iyakutti and Alagarsamy (2011) state that the design phase in banking involves translating the business requirements into functional requirements, which enables technical design and construction of the system according to the proposed solution. The authors recommend proceeding with this phase once the business requirement documentation has been developed and the completed technology assessments have been conducted. They emphasise the importance of having detailed specifications that contain all necessary information in order to construct the new system accurately.

They recommend validating and verifying the phase by reviewing the technical design, reviewing estimates on the project plan, and reviewing the system test plans. Through their research, Tanrikulu and Ozcer (2011) found that software development problems common to banks related to the design phase are lack of reviews, tests, problem reporting and resolution, documentation development and maintenance, test requirements development, training, risk management and proper processes for change management.

Iyakutti and Alagarsamy (2011) state that the development phase in banking involves development of the software and infrastructure, and the development and execution of unit testing. They recommend proceeding with this phase once the functional specification, technical design, system test plans, and coding and infrastructure standards have been developed. The authors recommend that this phase should be validated and verified by reviewing the test plans and scripts, reviewing the project plan, reviewing the code and infrastructure changes, and reviewing the unit test results.

Through their research, Tanrikulu and Ozcer (2011) found that software development problems common to banks related to the development phase are lack of coding standards and procedures, software configuration plans, policies and procedures, documentation of roles and responsibilities, a detailed implementation management plan, documentation of software unit and system development, documentation of unit testing results, integration test plans and draft versions of user documentation.

Iyakutti and Alagarsamy (2011) suggest that the testing phase in banking should involve testing whether the system meets the specifications, whether the system is accepted by the business, (i.e. the end user), and finally whether the system will be sustainable in a production environment by performing regression testing. The authors recommend proceeding with this phase once the coding scripts, test plans, and functional and technical design of the system have been developed. They further recommend validating and verifying the phase by the test results, verification of security assessments, approval for implementation, and user readiness for implementation. Through their research, Tanrikulu and Ozcer (2011) found that software development problems common to banks related to the testing phase are the lack of: integration plans, documentation regarding problems during experienced during installation, test design specification documents and approval of tests performed.

Iyakutti and Alagarsamy (2011) state that the implementation phase in banking involves moving the system into a production environment after successful test results have been obtained. They recommend proceeding with this phase once positive test results for system testing, user acceptance testing and regression testing are obtained. They further recommend validating the phase by confirming that migration of the system is according to the configuration management details. Through their research, Tanrikulu and Ozcer (2011) found that software development problems common to banks related to the implementation phase include failure to operate the production environment according to standard operating procedures, and the lack of formal problem management procedures and documentation procedures. The authors further found that problems common to the banking industry and related to the maintenance phase are lack of: implementation plans, formal approval of project completion, post-operation reviews and process, inspection of documentation and design and process verification.

2.4 Systems Engineering Development Models

London (2012) states that various processes, known as development models, can be used in SE development and management. He explains that these processes describe the engineering process across the system's life cycle. He mentions the Waterfall, Spiral, and V-model as the most popular development models. Rather and Bhatnagar (2015) describe a development or process model as a process that describes aspects such as specification, design, validation and evolution.

Buede (2009) explains that similarities can be seen between these models although they exhibit different characteristics. He suggests that factors such as uncertainty in requirements, risk, maturity of technology and prioritisation should be considered when selecting a development model. He further suggests that it should be kept in mind that there is not a standard model that fits all situations and that the development models can be used in combination if needed.

2.4.1 The V-model

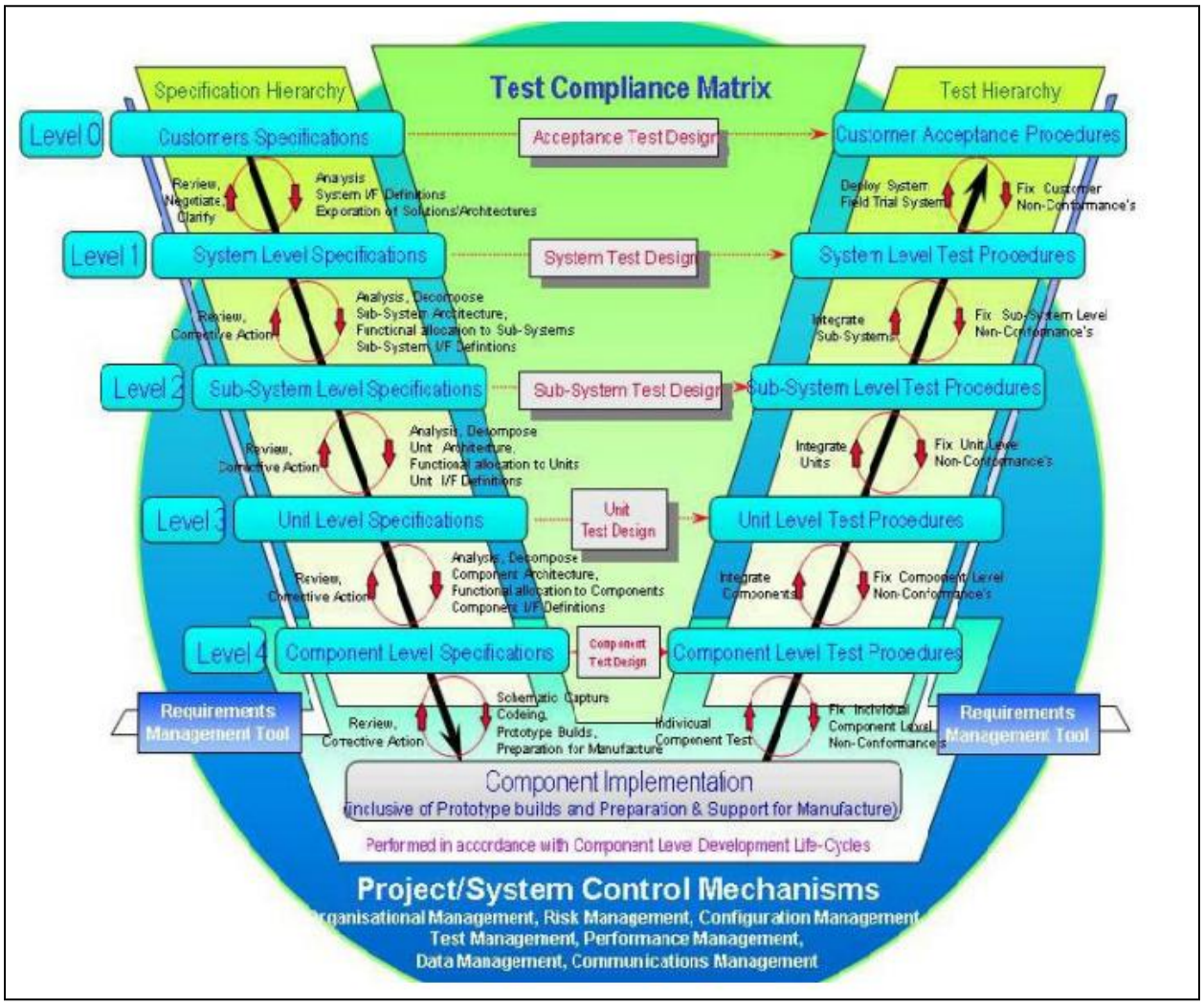


Figure 2.2 The V-model (Kasser, 2007)

Buede (2009) states that the V-model as represented in Figure 2.2 is one of the most popular development models. He describes the V-model as a sequential model in which software is designed on the left-hand (downhill) part of the model, and built and tested on the right-hand (uphill) part of the model. He states that the correspondence between the left and right hand activities are depicted by the lines across the middle of the V-model, which shows the levels from component testing at the bottom, integration and system testing, and acceptance testing at the top level.

The benefits of this development model include that it is focused on keeping stakeholders involved throughout the entire development process, concurrent opportunity and risk management, and verification and validation of each development activity leading to early problem resolution. Early problem resolution can save time and money as problems that are detected earlier in the development process are often easier and less expensive to fix (Buede, 2009). Mooz and Forsberg (2004) describe the V-model as being beneficial in terms of addressing software development complexity, decomposition, definitions, integration and verification.

Kasser (2007) warns that a shortcoming of the V-model is that although it allows for detecting defects, it does not allow for avoiding defects. When defects are identified and avoided at the outset, a project may experience greater cost saving and reduced risks. He suggests that defects should be kept in mind during the development phase so that these are not built into the system. He recommends that testers communicate the defects to the development team once they become aware of them. He further suggests that requirements should be updated to avoid defects to start with.

Rather and Bhatnagar (2015) state that for software development, the V-model necessitates that the requirements be clearly defined before the project starts at it is expensive to go back and implement changes once the software is implemented. They therefore suggest that it would not be the most appropriate model for projects where the requirements have a high risk of changing but suggest that it would be a good choice for critical projects. They state that the model emphasises risk analysis, allows for early production, and is easy to manage and understand.

However, they warn that it is a costly model that would not be suitable for projects that are small, long and ongoing, and object oriented. Munassar and Govardhan (2010) suggest that the rigidity of the model does not allow for enough flexibility and causes the adjustment of scope to

be difficult and expensive. They mention that an additional disadvantage is that the model does not describe or prescribe a clear way to resolve software defects.

2.4.2 The Waterfall model

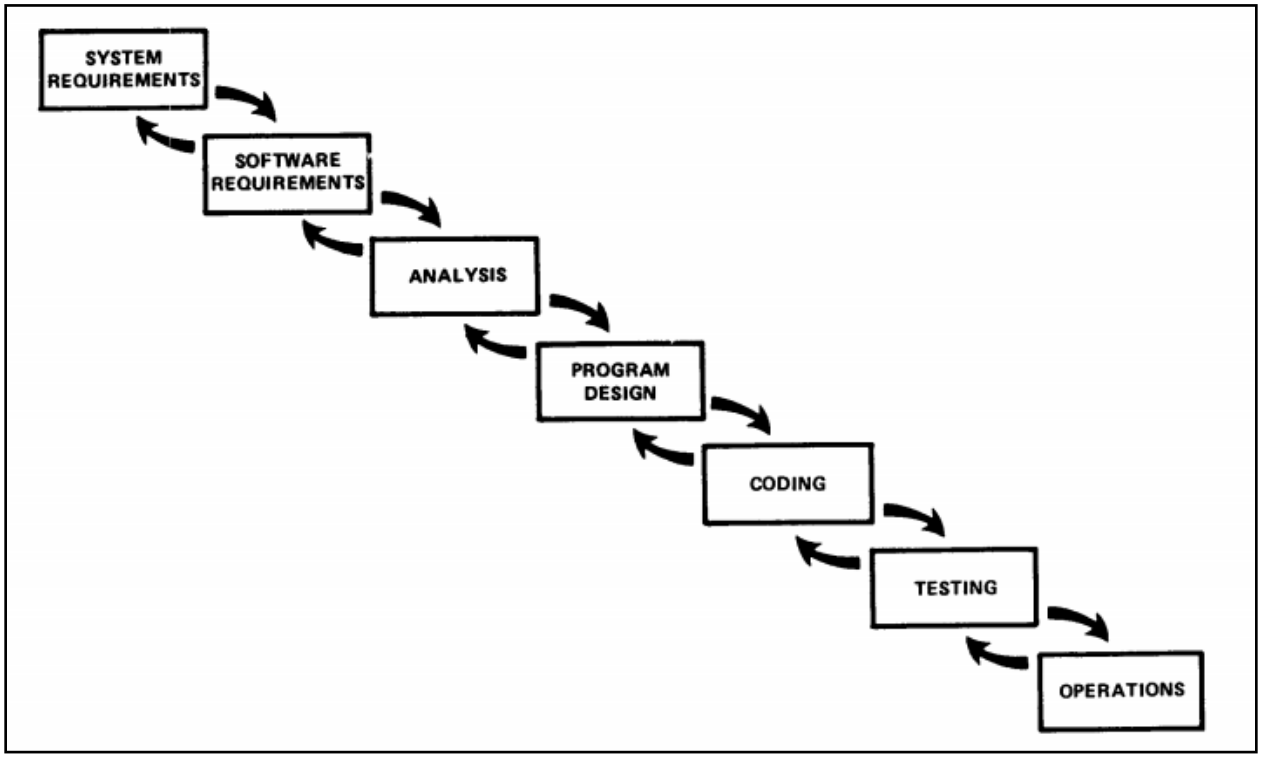


*Figure **2.3 The Waterfall** Model (Royce, 1988)*

Buede (2009) states that the waterfall model as represented in Figure 2.3 is one of the earlier developmental model designs in SE which is characterised by sequentially moving through the different life cycle phases. He describes the model as developing a system by moving from one phase to the sequential phase where iteration can only occur between the adjacent phases thus not allowing movement more than one phase forward or backward. Mooz and Forsberg (2004:4) assert: “The model promotes knowing the requirements before designing and designing before coding, etcetera.”

In terms of software development, Rather and Bhatnagar (2015) state that the Waterfall model necessitates that all the requirements be stated at the start of the project. The authors conclude that the model presents disadvantages such as problems remaining undiscovered until testing, unclear requirements, lack of risk management, difficulty in making changes and late delivery.

They state that although the model is easy to understand, it should be used in projects that are simple and have strict deadlines. Munassar and Govardhan (2010) contend that it is unrealistic to expect accurate requirements so early in a project and that it is costly and difficult to make changes to projects. Although the model allows for planning early stages, the software is delivered late in the project, which could delay the discovery of serious errors.

Buede (2009) states the design of the Waterfall model as a limitation as each step is not tested and verified before proceeding to the next which could result in detecting problems later in the development process. Kasser (2007) agrees that the design is limited and states that it might be necessary to go more than one step back to improve or adjust a previous step in the model.

2.4.3 The Spiral model

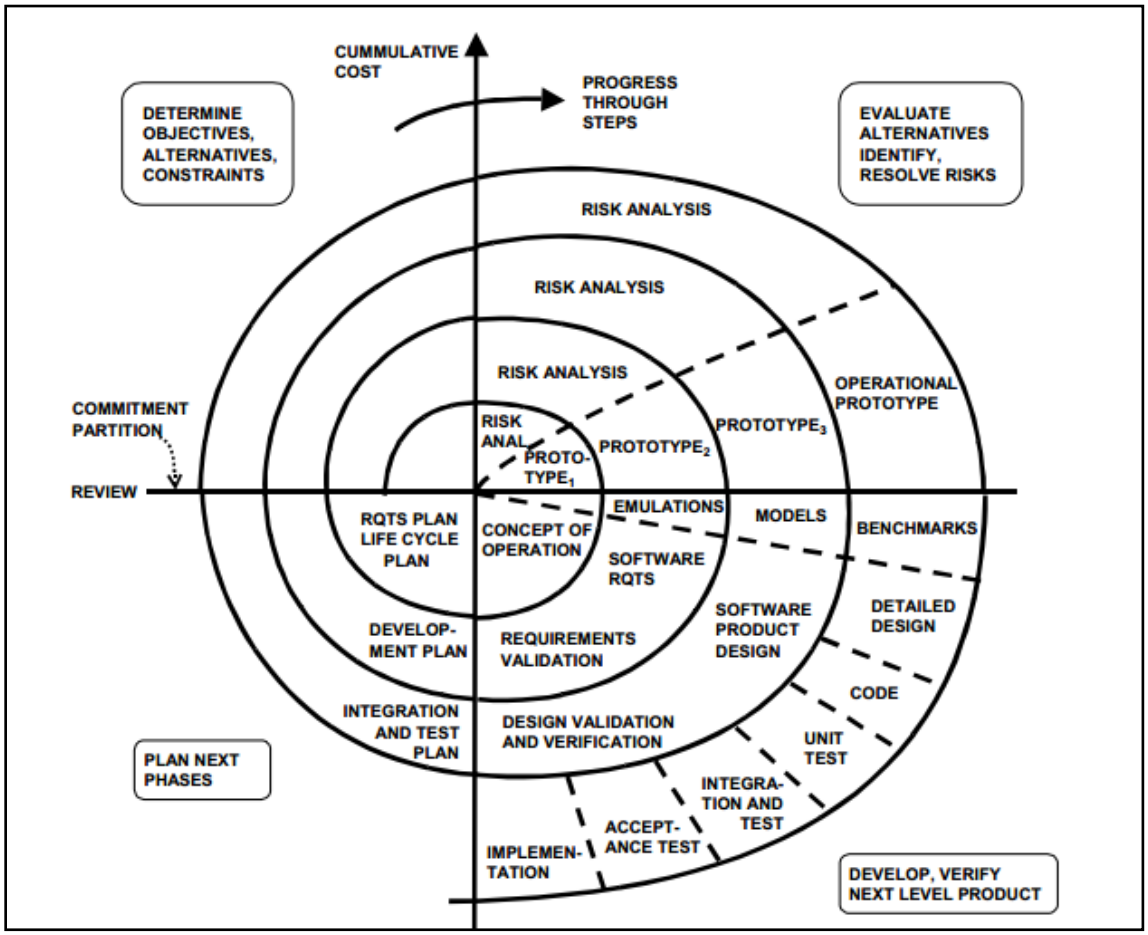


Figure 2.4 ***The Spiral Model (Boehm, 2000)***

Buede (2009) describes the Spiral model as represented in Figure 2.4 as a model that has a spiral-shaped design with four major sequential processes, namely (i) design, (ii) evaluation and risk analysis, (iii) development and testing, and (v) planning. He states that these processes will be repeated as needed until the final implemented product meets the requirements of the stakeholders.

Mooz and Forsberg (2004:4) contend:

> The model promotes resolving requirements, feasibility and operational risks prior to proceeding with traditional waterfall phases. The objective is to involve users and stakeholders in resolving recognized software development issues preceding

design. Although designed for software the model is also applicable to hardware and system development.

Boehm (2000) describes the Spiral model as a risk driven model that seeks to decrease risk throughout the development and implementation of the project and obtains agreement and commitment from the stakeholders on requirements throughout the life cycle. He explains that different processes and prototypes within the Spiral model are chosen for different projects according to the planning and risk analysis that was performed.

Buede (2009) suggests an advantage of this model is that it allows for different prototypes to be developed to determine which requirements will have the most impact on cost, schedule, and performance. Thus the benefits include improving requirements, eliminating nonviable options, avoiding rework of processes and providing early functionality. He suggests that another advantage is that it shortens the development life cycle by reducing the time between the stakeholders' requirements and the implementation of the final product. Thus risks such as change of requirements due to system changes can be prevented. Rather and Bhatnagar (2015) state that the Spiral model in software development allows for new prototypes to be developed continuously, reuse capabilities, improved productivity and elimination of errors in early stages of development.

A limitation of this model is that it looks at the risks separately instead of looking at the system as a whole from the beginning (Buede, 2009). Although the Spiral model allows for risk analysis in software development, it is a complex development model where the cost of risk analysis on large projects could be high. Another disadvantage as inadequate time and cost estimation (Rather & Bhatnagar, 2015). Munassar and Govardhan (2010) warn that by using this model the project's success is highly dependent upon risk analysis, which requires a high level of expertise in order to be conducted successfully.

2.5 Systems Engineering Principles

Hari et al. (2008) state that companies and organisations operate in a modern market where "products and systems are more complex, customers' budgets are limited, and competition is

tougher". The authors suggest that "in order to succeed in such a market, an organization must continuously improve the profitability of its products and services".

London (2012:15) describes SE as:

> …an interdisciplinary field that emerged as an effective way to manage complexity and change. It focuses on defining customer needs and required functionality early in the development cycle, and proceeding through design synthesis to system validation while considering the complete problem. Systems engineering is based on a holistic perspective of problems and design. Practitioners consider how systems fit into the larger context, how they impact it, and how they are influenced. Just as importantly, they consider how the interacting system components relate to each other.

One of the aims of SE is to ensure that the stakeholders' needs are met cost-effectively and timeously. SE also translates the stakeholders' needs into requirements and facilitates the selection and implementation of the best design from other alternatives. London (2012) continues that decision making is accomplished through the use of a disciplined and collaborative approach between system engineers and interdisciplinary teams.

Through his research, Honour (2004) found that SE improves cost, scheduling and quality within projects. He also found that the quality of SE plays a major role in the outcome of projects. London (2012) states that a combination of methods is used to accomplish SE objectives. These methods can also be referred to as SE principles, techniques, practices or concepts.

Buede (2009) asserts that a major principle of systems engineering is the consideration of the entire system at each activity throughout the life cycle. "Ignoring any part of the life cycle while engineering the system can lead to sufficiently negative consequences, including failure at the extreme" (Buede 2009:3). Sweeney et al. (2011) state that this principle involves formulating and refining operational, functional, and performance requirements, identifying and decomposing the system's functionality, implementing functionality into a feasible, useful product, verifying the system's requirements, functionality and implementation, and managing inherent operational, technical and programmatic risks.

Friedman and Sage (2003) support this view, proposing that the design activities in SE are performed from the viewpoint of the entire life cycle and not just the development phase. "A balanced blend of all methods, measurements, technologies, and processes shall be employed

in support of an effective life cycle" (Friedman & Sage, 2003:94). Funding support should be maintained throughout the project's life cycle and the development activities should recognise the total life cycle cost.

According to SE, the system life cycle has to be understood and areas of weakness should be identified by modelling the life cycle according to a systems development model (El-Sayar et al., 2013). College (2001) suggests that progress should be monitored, alternatives should be evaluated and selected, and data and decisions should be documented. The author further states that tools such as modelling, simulation, experimentation, and testing should be used to provide a rigorous quantitative basis for selecting performance, functional and design requirements and to evaluate alternative approaches to satisfy requirements and objectives.

The stakeholders' requirements should be analysed and defined in engineering terms and converted into specifications for the system and its components, segments and elements. "It is critical that this design process be broad in perspective so that nothing is left out and every contingency is considered" (Buede 2009:5). The specifications should be developed by considering the objectives of the stakeholders. The specifications should be detailed enough by specifying what the system must do, how well it must do it and how it should be verified and tested in order to adequately and correctly develop the system and its components (Buede, 2009).

College (2001:32) recommends that the requirements should be:

> …understandable, unambiguous, comprehensive, complete, and concise. Requirements analysis must clarify and define functional requirements and design constraints. Functional requirements define quantity (how many), quality (how good), coverage (how far), time lines (when and how long), and availability (how often). Design constraints define those factors that limit design flexibility, such as: environmental conditions or limits; defence against internal or external threats; and contract, customer or regulatory standards.

The author describes the process of functional analysis as:

> Functions are analyzed by decomposing higher-level functions identified through requirements analysis into lower-level functions. The performance requirements associated with the higher level are allocated to lower functions. The result is a description of the product or item in terms of what it does logically and in terms of the

> performance required. This description is often called the functional architecture of the product or item. Functional analysis and allocation allows for a better understanding of what the system has to do, in what ways it can do it, and to some extent, the priorities and conflicts associated with lower-level functions. It provides information essential to optimizing physical solutions.

Each function should have a corresponding requirement (College, 2001). Givens (2012) states that a core SE principle related to the requirements analysis phase is that problems should be clearly defined and a high level description of the functions that the system must perform should be generated. Sweeney et al. (2011) suggest that software requirements should be categorised, reviewed in order to identify duplicate and non-specific user needs, and assessed according to the user objectives in order to align with SE principles. The authors suggest that the functional analysis of the software system should include the preliminary design and planning activities by the developers, documenting the traceability between requirements and desired capabilities, relating the development plans into requirements, and detailing how the requirements should be implemented. El-Sayar et al. (2013) suggest that systems engineering requires that functional and non-functional requirements be gathered and analysed.

According to Friedman and Sage (2003), the following SE principles are related to the requirements phase:

- Requirements should flow from higher level to lower level requirements in a coherent and traceable manner
- Knowledge should be shared between resources and the user regarding all technical aspects of the system
- Users should be thoroughly involved in the development of the requirements

An important SE principle is the integration of the system. El-Sayar et al. (2013:7376) describe systems integration as “designing interfaces and bringing system elements together so they work as a whole”. They explain that integration “requires extensive communication and coordination. Interfaces between different subsystems, the main system and the customers must be designed. Subsystems should be defined to minimize the amount of information to be exchanged between the subsystems”.

College (2001:32) explains design as:

> …the process of defining the product or item in terms of the physical and software elements which together make up and define the item. The result is often referred to as the physical architecture. Each part must meet at least one functional requirement, and any part may support many functions. The physical architecture is the basic structure for generating the specifications and baselines.

The author states that the functional architecture should be revisited in order to verify that the physical design can perform the required functions and the required level of performance. Thus, it is possible to reconsider how the system will achieve its objective and permitting optimisation of the design. Friedman and Sage (2003) assert SE principles related to the design and development phases are:

- Establishing the system baseline architecture early in the systems development
- Involving technical issues, customer needs, political pressures and funding in the baseline architecture
- Performing judgment on issues by the development team and end user
- Designing the system in a logical and orderly manner according to the system functional architecture through functional decomposition and design traceability
- Sharing systems design responsibility between development team and end user
- Ensuring that each interface and integration step throughout the life cycle support total system functionality
- Ensuring that the system is integrated and interfaced with other systems
- Ensuring that all operational systems are compatible with each other

El-Sayar et al. (2013:7373) state that "it is required to view the enterprise, design, plan, implement, and govern the enterprise architecture". Buede (2009) suggests that integration should be planned during the design phase in a manner that simplifies verification and validation. He further suggests that likely alternatives to the integration order should be considered and the design should therefore be flexible enough to allow for these changes. El-Sayar et al. (2013) substantiates this opinion and states that proposed alternative solutions should be explored and assessed.

A core SE principle related to the design phase is that "alternative designs are created and evaluated based on multiple criteria, including performance, schedule, cost, and risk. Preferred options are modelled and evaluated, creating simulation data for analysis. Trials are ultimately run on the model system of choice" (Givens, 2012:68). The author explains that the preferred

alternative is then used to manage the system life cycle. Givens (2012:69) further explains that "the system is integrated into other systems with which it must interact. The outcome of effective integration is improved efficiency".

"For each application of the systems engineering process, the solution will be compared to the requirements. This part of the process is called the verification loop, or more commonly, Verification" (College, 2001:32). The author states that each requirement should be verifiable and the requirements documentation should detail the method of verification for each requirement.

El-Sayar et al. (2013:7376) contends that SE principles related to system verification and validation require "any system to verify and assess its performance, cost, and risk to ensure it is on track for fulfilling customer needs, also to verify and continuously assess system performance that is useful for feedback and re-evaluating the system". Friedman and Sage (2003) add that SE principles related to the testing phase are: testing every requirement, determining the success criteria and measures for testing early in the life cycle, involving users in the verification and validation of requirements and making the users the final approvers of the test outcomes.

Sweeney et al. (2011) suggest that SE applied to software testing requires that test cases should be clearly defined by describing and documenting each test case and mapping the test cases to the requirements. The test cases should be reviewed, modified and refined before test execution. They further suggest that a test results matrix should be developed and the results should be documented and archived for future reference.

Core SE principles related to implementation are that those operating the system are comfortable and knowledgeable regarding the functions of the system, the performance is assessed using ideally quantitative performance metrics, and the outputs of the system are observed in order to modify the system where necessary (Givens, 2012). Friedman and Sage (2003) mention the following SE principles are related to the implementation phase:

- During implementation testing the project team should maintain the appropriate technical capabilities to gather, analyse and recommend changes where necessary
- Implementation testing is performed by both the project team and end user, and reengineering is conducted where changes in design are necessary

- All data gathered during implementation testing is used for recommendations on future improvements

Sweeney et al. (2011) suggest that risks related to software development should be identified, planned, mitigated and tracked throughout the project across all aspects of the project. The risks should be identified by any team member and that all risks should be reviewed by the systems engineer and project manager in order to assess their relevance and probability. The authors suggest that a mitigation plan should be developed, executed and monitored to avoid the realisation of the risk. Once the risks have been found to be successfully mitigated, they could be retired. Friedman and Sage (2003) support this view, adding that according to SE, risk should identified, prioritised and mitigated early at every level of detail at all levels in the life cycle. According to College (2001:33), "risk management, configuration management, data management and performance-based progress measurement" should be conducted in order to ensure that:

- Alternative solutions "are made only after evaluating the impact on system effectiveness, life cycle resources, risk, and customer requirements"
- Traceability from inputs to outputs is maintained
- "Schedules for development and delivery are mutually supportive"
- "Required technical disciplines are integrated into the systems engineering effort"
- "Impacts of customer requirements on resulting functional and performance requirements are examined for validity, consistency, desirability, and attainability"
- "Product and process design requirements are directly traceable to the functional and performance requirements they were designed to fulfil, and vice versa"

Anderson and Nolte (n.d.:2) recommend that risk management in SE should address six elements:

1. "Identification of the potential sources of risk, and their drivers"
2. "Quantification of risks, including the probability of occurrence and seriousness of impact, and an assessment of their impacts on cost (including life-cycle costs), schedule, and performance"
3. "Determination of the sensitivity of these risks to program assumptions, and the degree of correlation among the risks"
4. "Definition and evaluation of alternatives to mitigate risks"

5. “Assurance that risk is factored into decisions on program objectives and design alternative analysis”
6. “Tracking risk items to ensure mitigation plans are effective, the potential impact on the program does not increase, and identify when risks become realized and become impediments to achievement of program goals”

An SE principle worth noting here is to constantly involve people from different backgrounds throughout the entire development process. Buede (2009:14) states that “in order to accomplish this difficult job of engineering a system, people with many different specialties must be involved on the systems engineering team”. He suggests that resources from the various technical and management areas as well the final users of the system should be involved throughout the entire life cycle.

2.6 Framework for Comparative Analysis

In order to perform the comparative analysis provided in Chapter 4, the major principles of SE as well as recommendations in the literature review that assist in the achievement of these principles are used as a baseline to compare to the current SDLC at the case site. These major principles are:

- Consideration of the entire system during each activity within the SDLC
- Definition and analysis of requirements that will achieve the users’ objectives
- Optimal design and integration of the system
- Optimal verification and validation of the system
- Intensive risk management during the entire SDLC
- Involvement of resources and users from all relevant disciplines throughout the entire SDLC

The recommendations obtained through the literature review in order to achieve these principles are tabulated and presented in Table 2.1.

Systems Engineering Principle	**Recommendations to Achieve Principle**	**Source**
Consideration of the entire system during each activity within the SDLC	Formulating, and refining operational, functional, and performance requirements	Sweeney et al. (2011)
	Identifying and decomposing the system's functionality	Sweeney et al. (2011)
	Implementing functionality into a feasible useful product	Sweeney et al. (2011)
	Verifying the system's requirements, functionality and implementation	Sweeney et al. (2011)
	Managing inherent operational, technical and programmatic risks	Sweeney et al. (2011)
	Performing design activities from the viewpoint of the entire life cycle	Friedman and Sage (2003)
	Using a balanced blend of methods, measurements, technologies, and processes that support the entire life cycle	Friedman and Sage (2003)
	Maintaining funding support throughout the life cycle	Friedman and Sage (2003)
	Ensuring development activities recognise the total life cycle	Friedman and Sage (2003)

	costs	
	Gaining a thorough understanding of the system's life cycle	El-Sayar et al. (2013)
	Understanding weaknesses by modelling the life cycle according to an SE development model	El-Sayar et al. (2013)
	Monitoring project progress	College (2001)
	Evaluating and selecting alternatives	College (2001)
	Documenting data and decisions	College (2001)
	Ensuring performance, functional, and design requirements, and alternative approaches satisfy requirements selected based on quantitative approaches	College (2001)
Detailed definition and analysis of requirements that will achieve the users' objectives	Obtaining a clear description of the environment wherein the software will operate	Hijazi et al. (2014); Iyakutti and Alagarsamy (2011)
	Identifying all the clients and collecting the raw requirements and objectives (functional and non-functional) from all points of view through observing and interviewing	Swarnalatha et al. (2014); Sommerville (2006) cited in Hijazi et al. (2014);

		Chomal and Saini (2014)
	Scheduling meetings with users and resources to obtain, analyse and review requirements	Chomal and Saini (2014); Iyakutti and Alagarsamy
	Developing standards and constraints in order to ensure common understanding	Swarnalatha et al. (2014); Hijazi et al. (2014)
	Defining problems clearly	Hijazi et al. (2014); Swarnalatha et al. (2014); Chomal and Saini (2014)
	Defining and analysing requirements in engineering terms and converting them into specifications for the system, its components, segments, and elements	Hijazi et al. (2014); Buede (2009)
	Analysing requirements by comparing it to user or business requirements or objectives	Swarnalatha et al. (2014); Iyakutti and Alagarsamy (2011); Buede (2009)
	Quantifying performance and interface requirements	Iyakutti and Alagarsamy (2011)
	Prioritising, categorising, reviewing, and assessing requirements according to users' objectives	Swarnalatha et al. (2014); Sweeney et al. (2011)
	Organising and defining requirements according to a hierarchy	Swarnalatha et al. (2014);

	from high level requirements that address business requirements to low level requirements that address component requirements in a coherent and traceable manner	Friedman and Sage (2003)
	Performing dynamic allocation by assigning the functional and non-functional requirements to the relevant system elements	Swarnalatha et al. (2014)
	Ensuring that requirements are clear, detailed, understandable, unambiguous, comprehensive, complete and accurate	Hijazi et al. (2014); Swarnalatha et al. (2014); Sweis (2015); Khan et al. (2014); Chomal and Saini (2014); College (2001); Givens (2012)
	Ensuring that requirements specify what the system must do, how well it must do it, and how the system should be verified and validated	Iyakutti and Alagarsamy (2011); Buede (2009); Givens (2012)
	Defining and clarifying functional requirements and design constraints	College (2001); Givens (2012)
	Performing functional analysis and preliminary design and planning activities by developers, documenting traceability between requirements and desired capabilities, relating development plans into requirements, and detailing how	Sweeney et al. (2011)

	requirements should be implemented	
	Developing requirements for each function	College (2001)
	Placing equal importance on functional and non-functional aspects	Hijazi et al. (2014); El-Sayar et al. (2013)
	Resolving conflicting user requirements by choosing the most relevant requirements that will achieve the users' objectives	Hijazi et al. (2014)
	Validating and verifying requirements by reviewing the requirements with clients, prototyping according to requirements and comparing system documentation to clients' requirements and objectives	Swarnalatha et al. (2014); Sommerville (2006) cited in Hijazi et al. (2014); Li (1990)
	Negotiating the requirements with resources and end users	Chomal and Saini (2014); Iyakutti and Alagarsamy (2011)
	Performing software requirements management by keeping track of and documenting all interrelationships and dependencies of software requirements changes	Swarnalatha et al. (2014)
	Thoroughly involving developers, end users and other necessary resources during requirements development	Hijazi et al. (2014); Friedman and Sage (2003)

	Documenting requirements for future reference and ensuring that these are consistent with requirements	Swarnalatha et al. (2014); Sommerville (2006) cited in Hijazi et al. (2014); Iyakutti and Alagarsamy (2011)
	Analysing cost and benefits	Iyakutti and Alagarsamy (2011)
	Reviewing preliminary project plans	Iyakutti and Alagarsamy (2011)
	Reviewing risks and contingency plans	Iyakutti and Alagarsamy (2011)
	Ensuring there is a formal process to track and control changes to specifications	Tanrikulu and Ozcer (2011)
	Ensuring there is knowledge sharing between resources and users regarding technical aspects of system	Khan et al. (2014); Friedman and Sage (2003)
	Providing appropriate documentation on processes and past project successes and failures	Hijazi et al. (2014); Tanrikulu and Ozcer (2011); Sweeney et al. (2011)

Optimal design of the system	Ensuring extensive communication and coordination between resources and users	El-Sayar et al. (2013)
	Creating and evaluating proposed alternative designs (solutions) based on multiple criteria, including performance, schedule, cost and risk	Givens (2012)
	Modelling and evaluating preferred options and run trials	Givens (2012)
	Using preferred alternative design to manage the system life cycle	Givens (2012)
	Defining subsystems	El-Sayar et al. (2013)
	Proceeding with design once business requirement has been received and completed technology assessments have been conducted	Iyakutti and Alagarsamy (2011)
	Ensuring design is broad in perspective and every contingency is considered	Buede (2009)
	Designing the system in a logical and orderly manner according to the system functional architecture	Friedman and Sage (2003)
	Choosing architectural design method and programming	Hijazi et al. (2014);

	language early in the design phase and according to the project's need	Friedman and Sage (2003)
	Revisiting functional architecture in order to verify that the design can perform the required functions and the required level of performance	College (2001)
	Involving technical issues, customer needs, political pressures and funding in the architectural design	Friedman and Sage (2003)
	Ensuring flexible architecture	Hijazi et al. (2014)
	Ensuring design is as simple as possible and understandable	Hijazi et al. (2014); Khan et al. (2014)
	Providing complete, clear and consistent documentation of the design process free from unnecessary information	Hijazi et al. (2014); Tanrikulu and Ozcer (2011)
	Performing judgment on issues and sharing systems design responsibility between the development team and end user	Friedman and Sage (2003)
	Ensuring maintenance of design document and specifications	Hijazi et al. (2014); Tanrikulu and Ozcer (2011)
	Accurately estimating available reusable components during	Hijazi et al. (2014)

	requirements analysis	
	Validating and verifying design by comparing to requirements documentation, reviewing technical design, reviewing estimates on the project plan and reviewing system test plans	Iyakutti and Alagarsamy (2011)
	Testing requirements development	Tanrikulu and Ozcer (2011)
	Setting up integration plan in a manner that simplifies verification and validation	Buede (2009)
	Exploring and assessing likely alternatives to the integration order and ensuring design is flexible enough to allow for changes	Buede (2009); El-Sayar et al. (2013)
Optimal integration and development of the system	Developing the system once functional specification, technical design, system test plans and coding and infrastructure standards have been developed	Iyakutti and Alagarsamy (2011)
	Correcting system decomposition and ensuring well-defined components and requirements	Hijazi et al. (2014)
	Ensuring each interface and integration step throughout the life	Friedman and Sage (2003)

	cycle supports total system functionality	
	Ensuring that the system is integrated and interfaced between different subsystems, systems, the main system and the customers	El-Sayar et al. (2013); Friedman and Sage (2003); Givens (2012)
	Ensuring all operational systems are compatible with each other	Friedman and Sage (2003)
	Excluding unnecessary specification of modules processing	Hijazi et al. (2014)
	Providing well defined functional definitions	Hijazi et al. (2014)
	Using one developer if possible	Hijazi et al. (2014)
	Following coding standards and best practices as well as good engineering standards	Hijazi et al. (2014); Chomal and Saini (2014); Tanrikulu and Ozcer (2011)
	Avoiding repetitive code	Hijazi et al. (2014)
	Ensuring reviewers can understand the code	Hijazi et al. (2014)
	Reusing components where necessary	Hijazi et al. (2014)

	Using experienced programmers	Hijazi et al. (2014)
	Ensuring good quality compilers and debuggers	Hijazi et al. (2014)
	Understanding of new technology before development	Hijazi et al. (2014)
	Performing integration incrementally according to the structure of the system	Hijazi et al. (2014); Li (1990)
	Ensuring correct versions and correct components are used for integration	Hijazi et al. (2014)
	Performing integration testing after each integration step	Li (1990)
	Validating and verifying the development phase by reviewing test plans and scripts, reviewing the project plan, reviewing the code and infrastructure changes, and reviewing the unit test results	Iyakutti and Alagarsamy (2011)
	Documenting software unit and system development	Hijazi et al. (2014); Tanrikulu and Ozcer (2011)
	Documenting roles and responsibilities	Tanrikulu and Ozcer

		(2011)
	Documenting unit testing results and integration test plans	Hijazi et al. (2014); Tanrikulu and Ozcer (2011)
Optimal verification and validation of the system	Proceeding with testing phase once coding scripts, test plans, and functional and technical design of the system has been developed	Iyakutti and Alagarsamy (2011)
	Ensuring that each requirement is verifiable, tested, and that the requirements documentation details the method of verification for each requirement	Hijazi et al. (2014); College (2001); Friedman and Sage (2003)
	Determining the success criteria and measures for testing early in the life cycle	Friedman and Sage (2003)
	Involving users in the verification and validation of requirements	Hijazi et al. (2014); Friedman and Sage (2003)
	Making the users the final approvers of the test outcomes	Friedman and Sage (2003)
	Setting up the test design specifications document	Tanrikulu and Ozcer (2011)
	Ensuring unit testing is performed during development phase	Li (1990); Yoon (2013)

	by the developer that coded the software	
	Using complete automated testing tools and appropriate testing techniques	Hijazi et al. (2014); Yoon (2013)
	Ensuring development of formal well-understood testing process	Hijazi et al. (2014)
	Documenting test cases for future use	Hijazi et al. (2014)
	Ensuring adequate regression testing by selecting all relevant test cases	Hijazi et al. (2014)
	Ensuring integration testing performed according to the structure of the system	Li (1990)
	Clearly defining test cases by describing and mapping each test case to the requirements	Li (1990); Sweeney et al. (2011)
	Reviewing, modifying, and refining test cases before test execution	Sweeney et al. (2011)
	Developing test results matrix	Sweeney et al. (2011)

	Documenting and archiving test results for future reference	Sweeney et al. (2011)
	Testing cases and data based upon system specifications	Li (1990)
	Using real-life data	Li (1990)
	Using qualified testing team	Hijazi et al. (2014); Chomal and Saini (2014)
	Ensuring sufficient number of testing resources	Hijazi et al. (2014); Chomal and Saini (2014)
	Allowing sufficient time for testers to test the entire system	Hijazi et al. (2014)
	Using latest verified requirements to test	Hijazi et al. (2014)
	Ensuring performance, stress and load testing is performed	Hijazi et al. (2014)
	Verifying and validating the testing phase performed by checking test results, verification of security assessments, approval for implementation, and user readiness for implementation	Iyakutti and Alagarsamy (2011)
	Ensuring formal approval of tests is performed	Tanrikulu and Ozcer (2011)

	Ensuring acceptance testing is performed by end users with the help of development team	Li (1990)
	Receiving formal acceptance of the system	Li (1990); Khan et al. (2014)
	Ensuring automated unit and regression testing	Parvez (2012)
	Updating regression test pack with latest test cases	Parvez (2012)
	Involving programmers with the design of unit test cases	Yoon (2013)
Successful implementation of the system	Proceeding with implementation phase once positive test results for system testing, acceptance testing, and regression testing have been obtained	Iyakutti and Alagarsamy (2011)
	Developing detailed implementation management plan	Tanrikulu and Ozcer (2011)
	Operating system according to standard operating procedures, formal problem management procedures and documentation procedures	Tanrikulu and Ozcer (2011)
	Ensuring that those operating the system are knowledgeable and comfortable regarding the functions of the system	Hijazi et al. (2014); Givens (2012)

	Assessing performance of system using quantitative metrics	Givens (2012)
	Observing outputs of system in order to modify system where necessary	Givens (2012)
	Ensuring project team maintains the appropriate technical capabilities to gather, analyse, and recommend changes where necessary	Friedman and Sage (2003)
	Ensuring implementation testing is conducted by both the project team and end user	Friedman and Sage (2003)
	Conducting reengineering where changes in design are necessary	Friedman and Sage (2003)
	Using all data gathered during implementation testing for recommendations on future improvements	Friedman and Sage (2003)
	Ensuring documentation of problems experienced during implementation	Tanrikulu and Ozcer (2011)
	Validating phase by confirming migration of the system is according to configuration management details	Iyakutti and Alagarsamy (2011)

	Obtaining formal approval of project completion	Tanrikulu and Ozcer (2011)
	Providing post-operation review process established	Tanrikulu and Ozcer (2011)
	Ensuring design and process verification	Tanrikulu and Ozcer (2011)
	Inspecting documentation	Tanrikulu and Ozcer (2011)
Intensive risk management during the entire SDLC	Creating work breakdown, clearly defining and assigning duties and responsibilities to resources over time	Chomal and Saini (2014)
	Understanding project complexities and choosing resources accordingly in order to have a clear project scope	Hijazi et al. (2014); Khan et al. (2014); Patil and Yogi (2011)
	Discussing project with experts	Patil and Yogi (2011)
	Learning from past experiences	Patil and Yogi (2011)
	Conducting assessments and reviews conducted early in the life cycle to mitigate risks	Iyakutti and Alagarsamy (2011)
	Identifying, listing, analysing, prioritising, mitigating and	Patil and Yogi (2011);

	resolving risks and their drivers early in the project and throughout the project	Chomal and Saini (2014); Sweeney et al. (2011); Friedman and Sage (2003); Anderson and Nolte (n.d.)
	Retiring risks once successfully mitigated	Sweeney et al. (2011); Friedman and Sage (2003)
	Quantifying risks, including probability of occurrence, seriousness of impact, and assessing impact on cost, schedule, and performance	Anderson and Nolte (n.d.)
	Determining sensitivity of risks to program assumptions and the degree of correlation among risks	Anderson and Nolte (n.d.)
	Defining and evaluating alternatives to mitigate risks	Anderson and Nolte (n.d.)
	Factoring risk into decisions on program objectives and designing alternative analysis	Anderson and Nolte (n.d.)
	Tracking risks to ensure mitigation plans are effective and the potential impact on the project does not increase	Sweeney et al. (2011); Anderson and Nolte (n.d.)
	Identifying when risks become realised and become impediments to achievement of program goals	Anderson and Nolte (n.d.)

	Ensuring risks are identified by any team members	Sweeney et al. (2011)
	Ensuring risks are reviewed by project manager in order to assess their relevance and probability	Sweeney et al. (2011)
	Monitoring and reporting on project performance throughout life cycle	College (2001)
	Adequately estimating project time, cost, scope and resources	Hijazi et al. (2014); Khan et al. (2014); Patil and Yogi (2011); Sweis (2015); Chomal and Saini (2014)
	Assessing all resource knowledge and capabilities before scheduling	Hijazi et al. (2014); Chomal and Saini (2014)
	Estimating schedule based on time spent on tasks and resource capabilities	Chomal and Saini (2014)
	Ensuring proper project planning and control in order to ensure realistic project schedule and budget	Hijazi et al. (2014); Khan et al. (2014); Chomal and Saini (2014)
	Ensuring good internal communication	Sweis (2015); Chomal and Saini (2014)

	Avoiding assumptions	Sweis (2015)
	Providing verification and validation of each activity within the SDLC	Hijazi et al. (2014); Li (1990); Tuteja and Dubey (2012); Khan and Khan (2014); Tanrikulu and Ozcer (2011)
	Ensuring end users are involved throughout the entire SDLC and development activities especially from the start of the project	Sommerville (2006) cited in Hijazi et al. (2014); Li (1990); Khan et al. (2014); Sweis (2015);Chomal and Saini (2014)
	Ensuring design changes made by end users result in timelines; revisiting requirements and project planning	College (2001)
	Monitoring project activities and milestones	Chomal and Saini (2014)
	Providing adequate training of resources	Khan et al. (2014); Sweis (2015); Chomal and Saini (2014); Tanrikulu and Ozcer (2011)
	Arranging regular planned meetings with all resources and end users to manage and agree on expectations, obtaining commitment from all involved and discussing results and	Khan et al. (2014); Chomal and Saini (2014); Iyakutti and Alagarsamy (2011)

	status of the project	
	Analysing requirements and resources to ensure realistic project expectations	Chomal and Saini (2014)
	Establishing proper processes for change management	Tanrikulu and Ozcer (2011)
	Providing a detailed implementation management plan	Tanrikulu and Ozcer (2011)
	Performing testing early in the requirements analysis phase	Tuteja and Dubey (2012); Khan and Khan (2014)
	Documenting resource knowledge	Sweis (2015)
Involvement of resources and users from all relevant disciplines throughout the entire SDLC	Encouraging knowledge sharing among team members	Friedman and Sage (2003); Khan et al. (2014); Hijazi et al. (2014); Chomal and Saini (2014)
	Ensuring good cooperation and communication among team members	Khan et al. (2014); Hijazi et al. (2014); Sweis (2015); Chomal and Saini (2014); El-Sayar (2013)

	Ensuring good motivation among team members	Khan et al. (2014)
	Involving resources from the various technical and management areas as well as the end users throughout the entire life cycle especially at the start of the project	Chomal and Saini (2014); Buede (2009)

Table 2.1 shows that the various SE principles and recommendations are interdependent. For example, involvement of all relevant disciplines assists risk management and a detailed requirements definition assists in successful implementation of the system.

2.7 Summary of the Literature Review

The literature review has covered all aspects of SE related to the development life cycle. SE principles and recommendations on how to align systems development life cycles to SE principles were analysed. This chapter forms the basis of the comparative analysis between the literature review and the current system presented in Chapter 4.

CHAPTER 3 – RESEARCH METHOD

"If we knew what it was we were doing, it would not be called research, would it?"

~ Albert Einstein

3.1 Introduction

There are various ways of gathering information from participants for the purposes of conducting research. In this chapter, the research methodology for collecting and organising appropriate research data is discussed and analysed.

The purpose of this chapter is to:

- Describe the overall research strategy to be used in the study and the roadmap followed to achieve the objectives presented in Chapter 1;
- Introduce the research instruments, namely face-to-face semi-structured interviews, visual sense making and process mapping;
- Discuss and justify the data analysis techniques used in the study; and
- Outline some limitations and ethical considerations considered in the study.

The primary objective of the study is to perform a comparative analysis between the current software SDLC and a recommended SDLC according SE principles. Following this, the study aims to identify areas where the current SDLC does not comply with SE principles and to provide recommendations on how to improve these areas.

The approach is to obtain information on the current system from the experts involved in the different phases of the SDLC and perform a comparison with SE as outlined in the literature review. Areas in the current SDLC that do not coincide with the recommendations in the literature are identified and recommendations are provided to improve these areas in terms of cost, schedule and quality.

3.2 Research Design

The study is an attempt to improve the current software SDLC in terms of cost, schedule and performance. Problems in the current SDLC experienced by management at a prominent South African financial institution include a long SDLC schedule, poor system verification, high resource capacity utilisation and high cost.

The research is primarily qualitative hence the study strategy is anchored on capturing the information from experts involved in the relevant areas.

The research project involved the following seven consecutive steps:

1. Gaining a detailed understanding of the current SDLC by conducting face-to-face semi-structure interviews with experts
2. Documenting the current software SDLC from information obtained during interviews
3. Mapping the current software SDLC process through visual sense-making from information obtained during interviews
4. Conducting an in-depth literature analysis at an academic level
5. Undertaking a comparative analysis between current software SDLC and the literature reviewed
6. Identifying inefficiencies in the current software SDLC according to the literature reviewed
7. Providing recommendations for improving the inefficiencies in the current software SDLC

3.3 Purpose of the Comparative Analysis

Walk (1998) states that the purpose of a comparative analysis methodology is to compare and contrast two or more things that can have similarities or commonalities yet can have crucial differences. Pickvance (2005) explains that a comparative analysis focuses on the explanation of differences and similarities. This research is aimed at seeking ways to improve the current SDLC by comparing the SDLC processes to systems engineering principles related to software systems development life cycles. The differences in the processes are analysed in order to identify opportunities for improvement.

The main objectives of the comparative analysis were to understand the shortfalls in the current software SDLC and recommend improvements that can be made to the current SDLC.

3.4 Research Instrument and Methodology

In order to elicit the knowledge and perspectives of experts in the various areas of the current software system regarding the current SDLC, qualitative, semi-structured, in-depth, face-to-face interviews are conducted.

3.4.1 The interview process

Strauss and Corbin (1990:17) describe qualitative research as “any kind of research that produces findings not arrived at by means of statistical procedures or other means of quantification”. The authors explain that qualitative research can be used to gain new perspectives, examine new phenomenon, and gain greater in-depth understanding of the topic – all of which might be difficult to obtain via quantitative methods. Therefore a qualitative approach for this study is chosen as this research methodology allowed the researcher to gain an in-depth knowledge of the topic based on industry expert experiences and knowledge (Patton, 1987). Wengraf (2001:6) defines depth as gaining “a sense of how the apparently straightforward is actually more complicated, of how the ‘surface appearances’ may be quite misleading about ‘depth realities’”.

During the interviews for this study, a qualitative approach allowed the researcher to be able to respond and interact with the participants, provide immediate feedback, interpret non-verbal communication, collect information on multiple levels simultaneously, and to gain and probe further for explanations (Lincoln & Guba, 1985). Wilkinson and Birmingham (2003) explain that interviews are used to obtain detailed information about a topic or subject.

Face-to-face interviews (as opposed to questionnaires) allowed the researcher to obtain more of an insight into the meaning and significance of what is happening. This is because the face-to-face interviews with each participant concluded only once the researcher had obtained sufficient

information. Additionally, face-to-face interviews provided constant interaction between the researcher and participants where the researcher's understanding was validated by the participant. More data and insight was therefore obtained during the interviews as any uncertainties and confusions were addressed (Wilkinson & Birmingham, 2003). However, Creswell (2007) mentions that when deciding to conduct one-on-one interviews, it is important to select participants that are not afraid to talk and share ideas in order to avoid obtaining insufficient inadequate information. This was considered during the participant selection process.

There are different types of interviews – namely, semi-structured, structured and unstructured interviews. Wilkinson and Birmingham (2003) explain that semi-structured interviews allow the researcher to direct the interview more closely, using the predetermined questions but allowing enough flexibility for the researcher to determine the flow of information. Therefore semi-structured interviews were chosen for this study, in order to plan the scope of questioning and therefore assist the participants in remaining on topic.

However, there was enough flexibility to allow for additional questions or a change in direction of conversation when needed. The researcher's knowledge of the subject and case site was limited, therefore it was often necessary to ask additional questions or have an interactive discussion with participants in order to gain a holistic in-depth view of the topic.

The researcher followed the approach as detailed in Figure 3.1 in conducting the interviews.

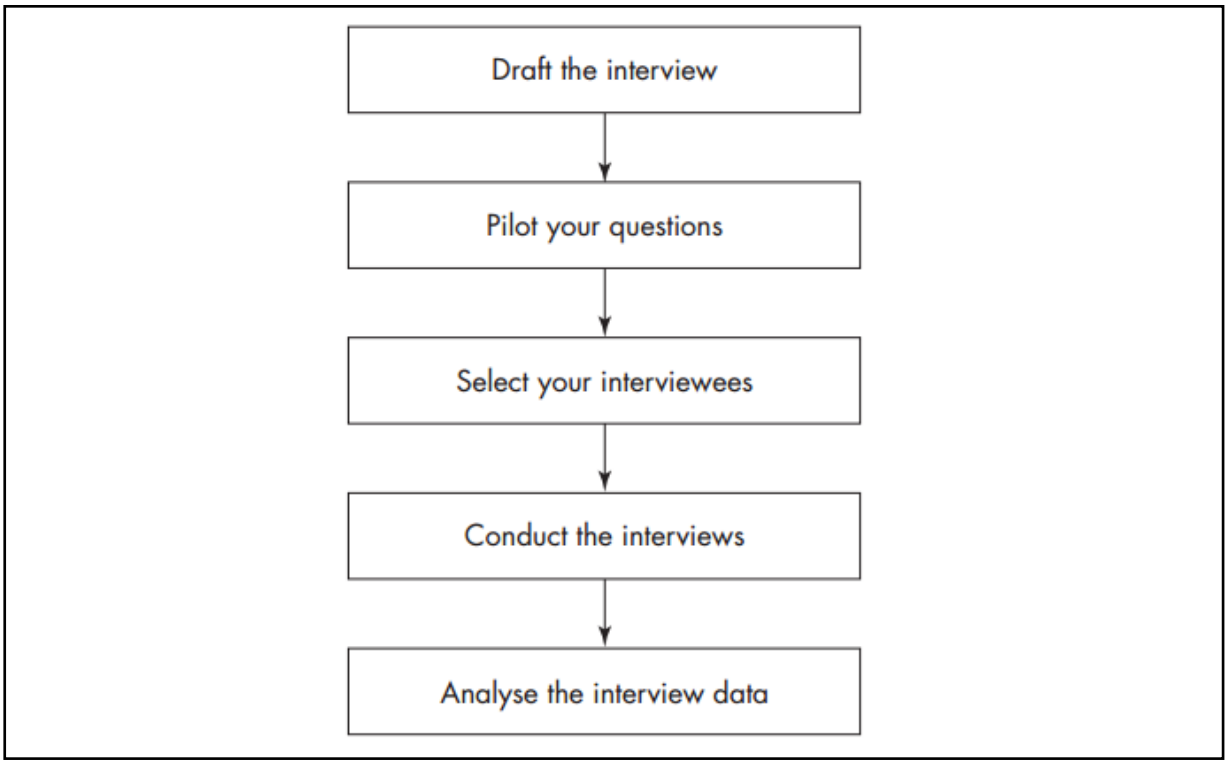


Figure 3.1 The Interview Approach (Wilkinson & Birmingham, 2003)

Step one: Draft the interview

Flick (2009:98) recommends "formulating research questions in concrete terms with the aim of clarifying what the field contacts are supposed to reveal". In order to draft the interview the following steps were followed:

Prepare themes or question areas. Wilkinson and Birmingham (2003:47) recommend the preparation of themes or question areas: "In any interview – structured, semi-structured, or unstructured – it is important for the interviewer to prepare a list of key questions to be covered so that important issues are not overlooked and the interview follows a logical progression." Themes are identified that relate to the different aspects of the system that require analysis namely, the structure of the system, the function of the system, the operation of the system, and strengths and weaknesses in the system. Open-ended questions are asked to ensure that the maximum amount of information is obtained. Irrelevant answers are disregarded. Creswell

(2007) suggests that open-ended questions should be designed to allow the participant to freely express his or her opinions. However, Flick (2009) suggests that although the openness should be applied to the questions, it is important to clearly formulate and define the research questions as to avoid having to analyse large amounts of data that are not relevant to the critical research question.

In order to understand the structure of the system, participants were asked to describe the particular area of the SDLC that they were responsible for, describe the processes involved, describe the chronological order of the processes, and describe the inputs and outputs for the different processes or stages. In gaining an understanding of the function and operation of the system, participants were asked to explain how each stage or process worked, describe the relevant inputs and outputs, and provide details of the resources, departments and timelines involved. Lastly, in order to understand the strengths and weaknesses in the system, participants were asked what they perceived the problems and successes of the system to be while taking into account the entire life cycle and how they thought the problems could be addressed.

Figure 3.2 illustrates how the interview questions were divided into the different themes.

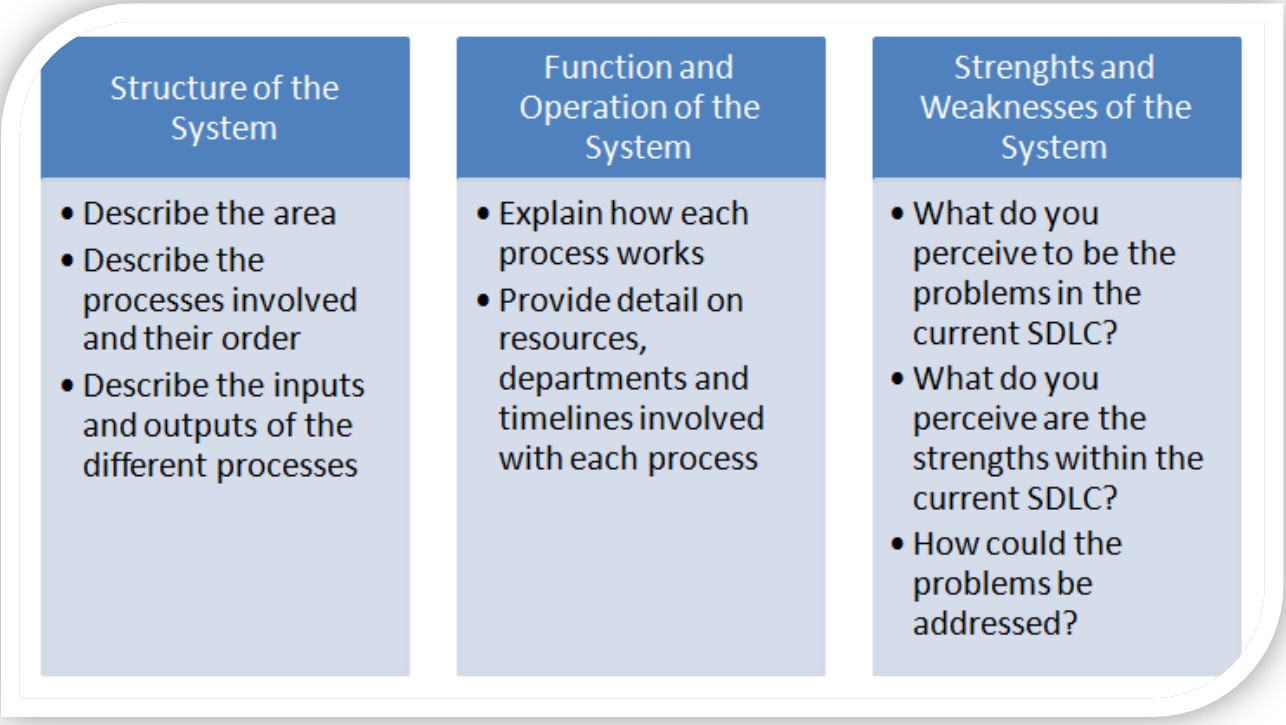


Figure 3.2 Determining Interview Themes for the Study

Clarify the number, type and sequence of questions. Wilkinson and Birmingham (2003) recommend to determine the number, type and sequence of questions and to ensure that the questions are necessary and clearly stated in order to gather as much information as possible. Flick (2009) recommends that the number of questions should be based on obtaining the important or necessary aspects of the research field. Therefore there is no recommended number, type or sequence of questions. The author suggests that the number, type, and sequence of questioning should be determined by how well it enables the researcher to answer the critical research questions.

The questions related to the different themes are reviewed in order to determine whether they are necessary, clearly stated as to avoid confusion for the participant, and to determine whether additional questions are necessary to understand the different aspects of the SDLC in detail.

Decide how interviews will be recorded. Creswell (2007) states that adequate recording procedures and equipment should be used when conducting one-on-one interviews. How the interviews will be recorded are finally determined at the drafting stage. Wilkinson and Birmingham (2003) state that audio recordings have been extensively used by researchers in the past. The authors state that an advantage of audio recordings is that they are able to be transcribed which provide more accurate and detailed information of the interviews as opposed to notes or summaries as they are records of every word that was spoken between the interviewer and interviewee.

It is therefore decided that interviews be audio-recorded and subsequently transcribed in order to capture all information during the interview. Video recordings are not chosen as the benefits that video recording provide such as visual cues, body language, and identification of who is speaking in a group environment are not necessary for the purposes of this study.

Step two: Determine pilot questions

Wilkinson and Birmingham (2003:52) state the importance of piloting interview questions: “Piloting is crucial. It assists in eliminating ambiguous questions as well as in generating useful feedback on the structure and flow of your intended interview. As with other research instruments, interview questions should be easy to understand.” Creswell (2007) agrees with this notion and suggests that the interview questions and procedures should be refined through pilot testing. This idea is confirmed by Flick (2009) who states that research questions are refined and reformulated as the research proceeds.

The interview questions were piloted on three participants working in various areas of the SDLC. These participants were not the experts with which the official research interviews are conducted. The responses of these participants were assessed in order to determine whether sufficient information would be obtained to understand the entire development life cycle. It was found that the feedback from the participants provided sufficient information in order to construct and understand the current SDLC. As the questions were open-ended and allowed for constant communication and feedback, it was possible to construct the processes of the SDLC accurately.

Step three: Selecting participants

Wilkinson and Birmingham (2003) state that because a tremendous amount of effort is involved in conducting interviews, the selection of the sample group must be representative of the data as well as sensible. The authors recommend that the central research question should guide the researcher in determining who to interview and how many interviews will be required. Leedy and Ormrod (2005) describe sampling as selecting a portion of the population that is representative of the entire population.

There are two main sampling methodologies used for selecting sample groups, namely random probability sampling and purposeful sampling. Patton (1990) explains that random probability sampling permits generalisation from the sample to the population it represents. Therefore, probability sampling often proves useful in quantitative studies, whereas purposeful sampling is focused on obtaining information rich cases regarding issues of central importance for in-depth study. Coyne (1997) describes purposeful sampling as selecting a sample based on the purpose or aim of the research.

Koerber and McMichael (2008) state that there are three major categories of sampling, which are convenience sampling, purposive sampling and theoretical sampling. They suggest that although convenience sampling is not ideal in all situations in can be applicable in some situations. The authors caution against over-generalising due to a narrow sample group. The authors explain that theoretical sampling differs from purposive sampling in that sampling develops along with the study and is not determined before the study takes place.

There was little benefit in seeking random sampling as the participants chosen were unable to provide the in-depth knowledge that was required (Cohen, et al., 2007). Purposeful sampling was chosen as the sampling methodology of choice because a thorough detailed understanding

of the current software SDLC processes was required. Convenience sampling would therefore not have been sufficient in obtaining broad in-depth knowledge on the subject matter. As it was necessary to select participants prior to the field work in order to ensure information accuracy, theoretical sampling was not considered.

In selecting a purposeful sampling methodology, it was important to avoid several pitfalls as outlined by Koerber and McMichael (2008):

- Selecting a sample that would not be large enough to represent the variation in the population. For example, selecting participants with similar knowledge and experience.
- Selecting a sample that would provide the results that suited researcher bias. This could be avoided by selecting participants from different organisations or areas within an organisation.
- Selecting a sample that would not achieve the purpose of the research.

From these pitfalls, it is evident that variability and applicability when selecting a purposeful sampling methodology are essential.

Patton (1990) as well as Miles and Huberman (1994) mention several purposeful sampling methodologies:

- **Extreme or deviant case sampling:** This approach focuses on cases that are unusual in some way. The lessons learnt from unusual cases can assist in improving typical cases.
- **Intensity sampling:** This method of sampling selects participants who experience the phenomenon in question intensely but not extremely.
- **Maximum variation sampling:** This method aims at capturing the central themes that emerge across participant variation. Participants who have unique experiences are selected, their variance in experience as well as common themes are analysed. Creswell (2007:126) states that maximum variation sampling is a common sampling technique that is often selected. “This approach is often selected because when a researcher maximizes differences at the beginning of a study, it increases the likelihood that the findings will reflect differences or different perspectives – an ideal in qualitative research.”
- **Homogeneous sampling:** This methodology selects a particular subgroup to obtain in-depth understanding.

- **Typical case sampling:** Through the use of key informants participants are identified whose knowledge and experience will provide typical cases of the phenomenon in question. This methodology does not provide a generalisation of all participants' views, but only that of the selected participants.
- **Stratified purposeful sampling:** This methodology combines typical case sampling with selecting participants that could provide information on below average and above average cases. The idea is to not only look at typical core cases, but also to investigate variation from the norm.
- **Critical case sampling:** Participants who could provide information that is critical or essential are selected. The information they provide is regarded as having the greatest impact on the development of knowledge.
- **Snowball or chain sampling:** Information-rich key informants are selected by asking resources associated with the phenomenon in question who the best possible resource is to speak to. As many resources are asked, key names will get mentioned repeatedly.
- **Criterion sampling:** Participants are selected based on predetermined criteria deemed important, such as participants who are involved in areas of weaknesses. These weaknesses can therefore be addressed in order to facilitate system improvements.
- **Theory based sampling:** Participants are selected based on important theoretical constructs.
- **Confirming and disconfirming cases:** Participants are selected who provide confirmation or disconfirmation on the information and knowledge obtained. These participants provide richness, depth, credibility on findings as well as boundaries on confirmed findings.
- **Opportunistic sampling:** The sample selection emerges during field work. As data is collected, new sources of information are identified.
- **Purposeful random sampling:** Participants are randomly selected according to the credibility that they provide to the information. This avoids questions as to why only certain participants were selected and does not allow generalisations to be made to the entire population.
- **Sampling politically important cases:** Participants are selected based on their political importance. Their political view could provide additional perspectives that could prove to be useful.
- **Convenience sampling:** Participants who are simple and easy to study are selected. Factors such as ease of access and low costs determine the sampling group.

For the purposes of this research a combination of intensity sampling, maximum variation sampling, typical case sampling, critical case sampling, chain sampling, criterion sampling, confirming and disconfirming cases, and opportunistic sampling was used. Intensity sampling was chosen as participants who were most involved within the particular area of the SDLC were selected.

Maximum variation sampling was chosen as participants were selected based on their experience in the unique areas within the SDLC. Variety in job description, duties and responsibilities and area of experience within the SDLC were factors in selecting participants. This was done to ensure that all aspects of the SDLC were covered.

Typical case sampling was chosen as the participants who were selected possessed knowledge on the general or typical SDLC processes. Critical case sampling was chosen as participants who were viewed as experts in their field and therefore presented the greatest possibility of obtaining accurate in-depth knowledge.

Chain sampling was chosen as participants were identified by asking all resources and project managers involved in the various areas of the SDLC who the most knowledgeable and experienced resource on the subject matter would be. Criterion sampling was chosen as participants who were involved in areas of the SDLC reported by management to have weaknesses in quality and schedule of project delivery were selected. Confirming and disconfirming cases were chosen as some participants who had the same job description as well as experience and knowledge of the same areas within the SDLC as other participants were selected. This was done to ensure that information obtained from participants was accurate. Finally, opportunistic sampling was chosen as some experts were identified as necessary once further information regarding the processes was obtained.

Creswell (2007:126) emphasises that both the sampling strategy and the sampling size are important considerations when selecting the sample. “One general guideline in qualitative research is not only to study a few sites or individuals but also to collect extensive detail about each site or individual studied.”

The central research question of this study is concerned with all aspects pertaining to the current software SDLC. Therefore, the experts responsible for the various areas in the SDLC were selected in order to ensure that variability and applicability would be maintained. This

strategy additionally ensured that extensive in-depth knowledge was obtained about all the various processes within the SDLC.

Step four: Conducting the interviews and data analysis

Wilkinson and Birmingham (2003) states that the setting where the interview is conducted forms an integral part of the interview and should not appear confrontational or intimidating. Creswell (2007:134) further suggests selecting a quiet place that is free from distractions and that allows adequate recording of the interview. The author further recommends having the interviewee complete a consent form prior to conducting the interview. He recommends that the researcher "go over the purpose of the study, the amount of time that will be needed to complete the interview, and plans for using the results from the interview".

Wilkinson and Birmingham (2003) recommend that the interviewer should sit alongside the interviewee with the recording device placed so as to not distract the interviewee. The authors suggest that note taking should be kept to a minimum so as to not distract the interviewee. They recommend that the interviewer introduce him or herself, explain the purpose of the study as well as its structure and format, how the data will be used, and how anonymity will be ensured.

They suggest that open-ended questions should be asked in order to encourage the interviewee to provide as much information as possible and suggest taking note of the interviewee providing comforting signs or acceptance cues such as head nodding, having an attentive posture, and keeping eye contact. The authors suggest that the interviewer restate part or all of the interviewee's responses in order to clarify what the interviewee has said and to encourage the interviewee to elaborate on what he or she has said.

They encourage the interviewer to use silence to encourage the interviewee to elaborate on his or her response. Creswell (2007) suggests that the interviewer should focus on being a good listener rather than a frequent speaker. He further recommends taking notes in addition to audio recordings in case the audio recordings do not work.

Finally, in concluding the interview, the interviewer should paraphrase what has been discussed in order to allow the interviewee to correct any statements or add information, thank the interviewee, and provide a summary of the interview and information (Wilkinson and Birmingham, 2003). Creswell (2007:135) further suggests that the researcher "write out the closing comments that thank the interviewee after the interview and request follow up information, if needed, from them".

The research was conducted at the financial institution in a meeting room where the interviewer and interviewee sat alongside each other. Introductions were made and information was provided on the purpose of the study, how the data was intended to be formatted, structured and used, and how anonymity would be kept. Open-ended questions relating to the SDLC were asked.

Data analysis took place during and after the interview. Wilkinson and Birmingham (2003) recommend that once all relevant data is gathered, data should be structured by grouping the information into themes, issues or concerns. The interviews were recorded with an audio recorder and process maps of the SDLC were drawn up by the interviewer during the interview as the interviewee provided information.

The process maps were drawn up by using sense making through the use of visual analytics. Shrinivasan (2010:1) explains the sense making process as follows:

> Visual analytics is the science of analytical reasoning facilitated by interactive visual interfaces. It involves representing information visually and allowing the human to directly interact with it, to gain insight, to draw conclusions, and to ultimately make better decisions. It aims to support the sense making process in which information is collected, organized and analysed to form new knowledge and inform further action.

The visual sense making process consists of two phases:

1. Capture and organise findings
2. Review and revise analysis

Capture and organise findings. Findings can be captured and organised by grouping themes from notes according to topics and sub-topics and creating casual relationships between notes (Shrinivasan, 2010). Denzin and Lincoln (1994) suggest that qualitative information should be sorted into meaningful categories.

The researcher recorded the information using audio recordings and documentation. Firstly, the interviewees were asked open-ended questions and the entire response and explanation of the particular area of the SDLC was documented and recorded. Responses were restated and silence was used where necessary. In instances where more information was necessary, the interviewees were asked to elaborate. Additional questions were asked where it was deemed necessary in order to fully understand the SDLC.

From the notes, the information was grouped into the different processes and functions within the particular area of the SDLC. Important aspects were captured and main processes were derived. The relationships between processes and chronological order of the different processes have been indicated using arrows.

Review and revise analysis. Shrinivasan (2010) suggests that visualisation states should be revisited via notes and recent steps should be reviewed to compare results or undo actions. The author additionally suggests that alternative solutions should be considered.

After the process maps were drawn up, each process was reviewed by revisiting the information gathered in the notes. The process maps were subsequently shown and explained to the interviewees in order to allow the interviewees to comment on the accuracy. Adjustments were made where necessary.

In concluding the interview, the interviewer paraphrased what had been said and discussed, thanked in the interviewees and provided a copy of the transcript.

The final stage of analysis involved reviewing the transcripts in order to capture the information related to the process maps and to verify the accuracy thereof. Data was grouped according to the main processes and information was documented in chronological order.

3.5 Data Analysis

Once the analysis for the current system had been reviewed and revised, an in-depth literature review was conducted. Literature on SE that pertains to all aspects of an SDLC was reviewed. Subsequently, a comparative analysis between the literature reviewed and the current SDLC was performed.

The comparative analysis was conducted as follows:

1. SE principles obtained through the literature review have been listed
2. Recommendations on aligning each life cycle phase to SE principles obtained through the literature review have been listed
3. Each of the current SDLC phases have been compared to the recommendations

4. Activities that did not coincide with the literature review recommendations have been indicated in the final column. These activities have been noted as the inefficiencies in the current system.

Finally, recommendations have been made to change the inefficiencies in the current system to coincide with the SE principles obtained from the literature review.

3.6 Reliability and Validity

Golafshani (2003) states that qualitative analysis results differ from quantitative analysis results due the fact that different participant views are formed from different paradigms. The concepts of reliability and validity need to be redefined in a qualitative study in order to reflect the various ways of obtaining the results. The author continues to explain that reliability and validity are not viewed separately in qualitative research, but rather terminology that encompasses both, such as quality, rigor and trustworthiness are used.

Golafshani (2003) suggests that, in order to determine the quality of the study, aspects such as credibility, neutrality or confirmability, consistency or dependability, and applicability or transferability should be considered to measure the quality of the study. The rigor of the study should be determined by exploring subjectivity, reflexivity, and the social interaction of interviewing. Finally, the author suggests that the trustworthiness of the study should be determined by establishing confidence in the findings.

The following aspects were considered in order to determine the reliability and validity of this study:

- ***Rigour in documentation:*** "Rigour in documentation ensures that there is a correlation between the steps of the research process and the study in question, commencing with the phenomenon of interest and following through to the recommendations and implications of practice" (Coughlan, et al., 2007:742). The study closely followed the research process recommended by Wilkinson and Birmingham (2003) and Shrinivasan (2010) in Section 3.3.

- ***Procedural rigour:*** "Procedural rigour refers to appropriate and precise data collection techniques and incorporates a reflective/critical component in order to reduce bias and misinterpretations" (Coughlan, et al., 2007:743). During the data collection process, the data was constantly reviewed by the researcher in order to determine if the participants' statements made logical sense relative to other statements and data received. Visual sense making was used to ensure that a holistic understanding of the information was obtained. The information was constantly restated and shown to encourage the interviewee to reflect on the information. The selection of interview questions was a threat to procedural rigour. The selection of interview questions could possibly have been limited without encompassing the aspects of the entire SDLC. To address this threat, the questions were piloted as detailed in Section 3.3.

- ***Ethical rigour:*** "Ethical rigour describes how confidentiality issues and the rights of participants are dealt with during the research process" (Coughlan, et al., 2007:743). Responses from the interviews conducted are treated completely confidentially. Consent forms are signed by all participants. Information regarding the participant is not documented or recorded. Any sensitive information is not transcribed. Information solely related to the current software SDLC is transcribed. Copies of the transcriptions are given to the participants.

- ***Credibility:*** "Credibility addresses the issue of whether there is consistency between the participants' views and the researcher's representation of them" (Coughlan, et al., 2007:743). The process as outlined for procedural rigour ensured that credibility was maintained during the research process. A possible threat to credibility was the selection of the sample group of participants. Participants' own experiences and knowledge were taken into account for data collection. Therefore, there was a risk that the information could be biased and limiting. To address this issue, experts that covered all areas of the software SDLC were selected for interviews. Open-ended questions were asked, silence was used, and information was restated to the participant in order to obtain as much information as possible. Answers were constantly compared to information received from other participants as well as the same participant in order to ensure that the information made logical sense.

- ***Dependability:*** Coughlan, et al. (2007:743) describe dependability as: “Dependability involves the researcher giving the reader sufficient information to determine how dependable the study and the researcher are. A study may be deemed auditable when another researcher can clearly follow the trail used by the investigator and potentially arrive at the same or comparable conclusions.” The authors suggest: “It is also necessary for each stage of the research to be traceable and clearly documented”. A detailed explanation of the research methodology, data analysis and results have been outlined in this research study.

- ***Transferability:*** “Transferability refers to whether or not findings can be applied outside the context of the study situation” (Coughlan, et al., 2007:743). In today’s banking and financial industry, decisioning is largely automated by the use of software. Software development across industries and companies follows a life cycle with similar characteristics and phases to the SDLC of this study. The findings of this study do not solely relate to credit scoring processes within financial institutions, but can be transferred to any industry or environment where software development takes place.

- ***Confirmability:*** “Confirmability requires the researcher to demonstrate how conclusions and interpretations have been reached” (Coughlan, et al., 2007:743). The reasoning behind the results and conclusions are detailed in Chapter 5 and Chapter 6.

- ***Goodness:*** “Goodness needs to be evident in the philosophical background and study design, providing explicit explanations regarding the study context, data collection and management and the interpretation and presentation process” (Coughlan, et al., 2007:743). The goodness of the study has been defended in Section 3.1 to Section 3.3 of this paper.

3.7 Limitations

The following limitations were found during this research:

- The information obtained of the current software system was limited to the knowledge and experience of the participants involved in this study. Lack of documentation on current processes within the current software SDLC caused the study to be highly reliant on the information obtained from interviews conducted with the participants. Critical information could therefore have been excluded.
- To an extent the research depended on the participants' and researcher's interpretation. This introduced a degree of bias into the study.
- SE literature is vast. Therefore it was difficult to select and refine information relevant to the study and this was done so subjectively.

To address these limitations the researcher ensured:

- The selection of the participant sample group covered all areas of the current software SDLC are encompassed. Expert participants responsible for the various areas of the current software SDLC were selected to optimise the chances of gathering adequate and detailed information.
- Information was recorded accurately.
- The logic of the information and own biases were continuously reflected on.
- Clarification was sought in areas that were seen to introduce bias.
- A thorough literature review was conducted and research questions were continually refined to focus the investigation.

3.8 Ethical Considerations

The 1948 UN Declaration of Human Rights, the 1996 Constitution of the Republic of South Africa, and the ethical issues in research as outlined by Leedy and Ormrod (2005) guided the ethical considerations in this study. In addition, the following measures were adhered to:

- Informed consent: expert participation in the interviews was voluntary

- Confidentiality and anonymity: the research study respected the participants' right to privacy and their contribution has remained anonymous throughout the study
- The University of Witwatersrand Ethics Policy for non-medical research

Ethics clearance was obtained through the School Ethics Committee with School Ethics Clearance number of MIAEC 044/15. Above all, professional membership and the respective code of ethics of governing body INCOSE has been honoured in this study.

3.9 Chapter Summary

This chapter has provided a detailed description of the research approach. Information on the current software SDLC was gathered by conducting interviews with expert participants. Data was analysed during the process of sense making through the use of visual analytics. Insight obtained from the in-depth literature review was applied and a comparative analysis was performed between the current software SDLC and the literature review was conducted. Areas of inefficiencies in the current software SDLC were identified that did not align with SE principles and recommendations have been made on how to improve these areas.

CHAPTER 4 – DATA ANALYSIS AND RESULTS

"It is a capital mistake to theorize before one has data."

~ Sherlock Holmes

4.1 Introduction

The research question, as stated in Chapter 1 is:

> ***How can the software SDLC be improved in terms of cost, schedule, and performance for a credit scoring system using systems engineering principles?***

In Chapter 3, the research methodology was discussed, which aimed to answer the research question by establishing the following:

1. Gaining a detailed understanding of the current SDLC by conducting face-to-face semi-structured interviews with experts
2. Documenting the current software SDLC from information obtained during interviews
3. Mapping the current software SDLC process through visual sense-making from information obtained during interviews
4. Gaining an in-depth literature analysis at academic level
5. Undertaking a comparative analysis between current software SDLC and recommendations obtained through the literature review
6. Identifying inefficiencies in the current software SDLC according to the literature reviewed
7. Providing recommendations on improving the inefficiencies in the current software SDLC based on information obtained in the literature review

This chapter contains the data analysis for the current software system as well as the results of the comparative analysis performed between the current system and the literature review as outlined in Chapter 2. The results presentation follows the approach shown in Figure 4.1.

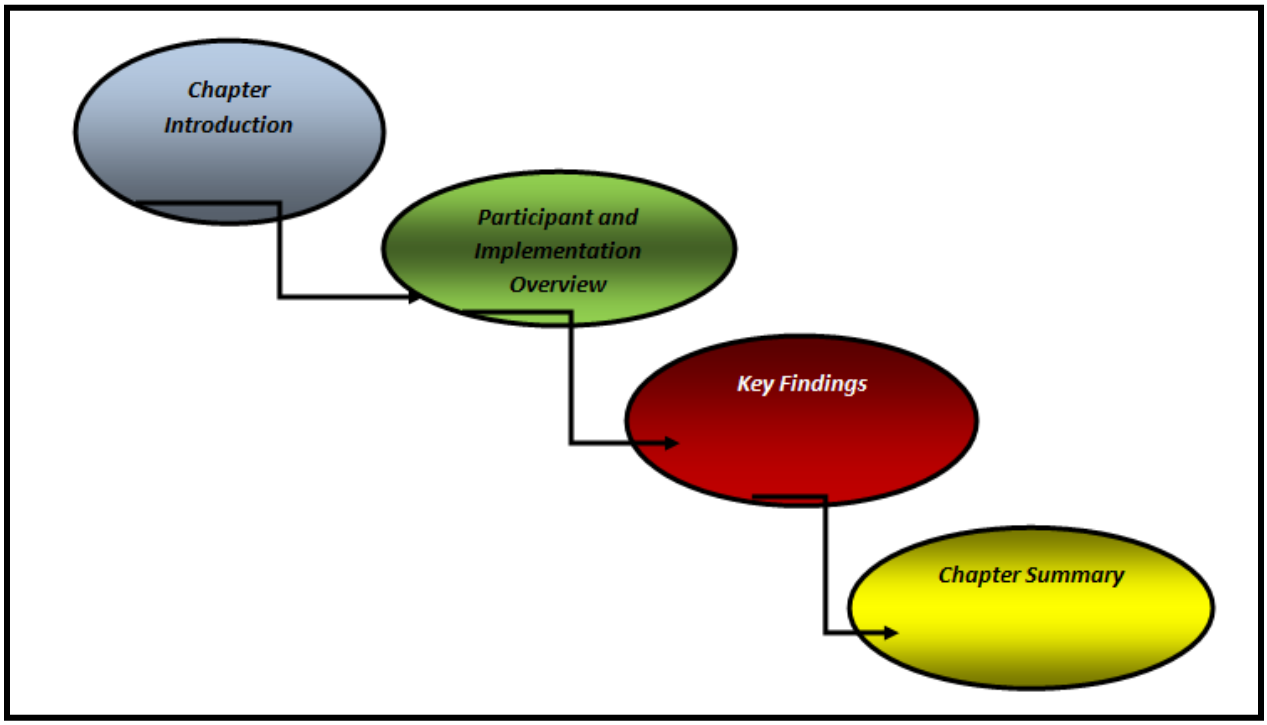


Figure 4.1 Results Road Map

4.2 Participant and Implementation Overview

Ten experts were identified to be key resources from whom a detailed understanding of the current system could be obtained. All 10 experts participated in the study. The participants were considered experts based on their seniority, level of involvement in the SDLC on a monthly basis, and years of experience in software development systems. The experts' knowledge covered the different phases of the SDLC in detail. The participants' roles in the current system are detailed in the Table 4.1.

Table 4.1 **Participant information**

Expert	Role	Phase of the SDLC covered	Number of years experience in SDLCs
Participant 1	*Project Manager*	*All SDLC phases*	*9 years*
Participant 2	*Developer*	*Testing phase*	*15 years*
Participant 3	*Quantitative Analyst*	*Requirements analysis phase, Design phase, Testing phase*	*15 years*
Participant 4	*Test Analyst*	*Testing phase*	*11 years*
Participant 5	*Systems Analyst*	*Requirements analysis phase, Design phase*	*25 years*
Participant 6	*Business Analyst*	*Requirements analysis phase, Design phase*	*9 years*
Participant 7	*Test Analyst*	*Requirements analysis phase, Testing phase*	*10 years*
Participant 8	*Test Manager*	*Testing phase*	*12 years*
Participant 9	*Business Unit Analyst*	*All SDLC phases*	*16 years*
Participant 10	*Project Manager*	*All SDLC phases*	*20 years*

4.3 The Current Software SDLC

This section contains information regarding the current software SDLC from interviews conducted with participants at a prominent South African financial institution. Open-ended questions were asked and each participant's entire response and explanation of the particular area of the SDLC was documented and recorded. Important aspects were captured and main processes were derived from the data. The information was grouped into the different processes and functions within the particular area of the SDLC. Information was collated and process maps were drawn up by documenting information from the different participants in chronological

order. Each process was reviewed by the participants and the interviewee. Adjustments were made where necessary. The information obtained from the various participants is detailed in Table 4.1. Interview transcripts are presented in Appendix D.

4.3.1 The credit scoring process for the case site

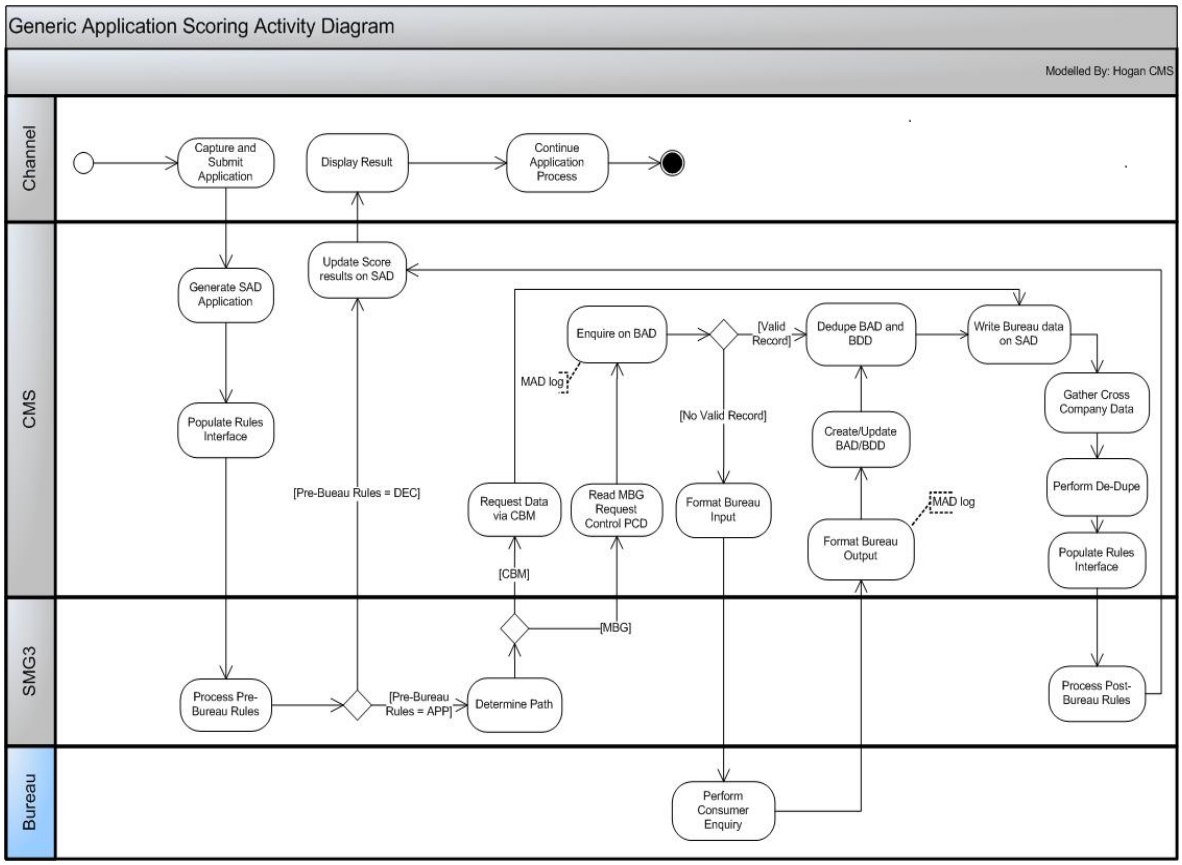


Figure 4.2 The Credit Scoring Process

Figure 4.2 represents the credit scoring process for the case site as illustrated by the researcher. Currently, when a customer submits a credit application, the application is captured by the respective banking channel and submitted to the customer management system (CMS), which generates the application on a single application database (SAD). CMS also populates the rules interface and then sends the data to the strategy design studio system (in this case the

Strategy Management Generation 3 or SMG3), which runs the customer's data through the pre-bureau rules. If the customer infringes any of the rules, the customer is declined, and the result is updated on the SAD by the CMS. The banking channel then displays the result and the customer is informed that he or she does not qualify for a credit loan.

However, if the customer passes all the pre-bureau rules, the customer's data from the credit bureau is requested via either the credit bureau mainframe (CBM) or mainframe bureau gateway (MBG). In the case of bureau data requested via CBM, the customer's SAD record is updated with this credit bureau data.

A request to the different product houses is subsequently performed (known as cross company call) in order to obtain any internal company credit history for the customer. Should the customer have internal credit information, the corresponding record from the bureau is deleted and only the internal record is kept. Subsequently the rules interface is populated by CMS where after SMG3 runs the customer's data through the applicable credit origination scorecard, limit and affordability calculations, and post-bureau rules in order to obtain the customer's score, final approval, and limit offered. These results are updated on SAD, the results are displayed and the customer is informed of the decision. The fulfilment process subsequently takes place.

Should the customer's credit bureau data be requested via MBG, an enquiry is performed to the bureau aggregations database (BAD), which stores all previous customer data received from the credit bureau. If the customer's credit bureau data does not appear on BAD, an enquiry is performed to the credit bureau in order to obtain the customer's credit bureau information. BAD is then updated with this customer's credit bureau data and the customer's SAD record is updated with this information. A request to the different product houses is subsequently performed (cross company call) in order to obtain any internal company credit history for the customer.

Should the customer have internal credit information, the corresponding record from the bureau is deleted and only the internal record kept. Subsequently the rules interface is populated by CMS, where after SMG3 runs the customer's data through the applicable credit origination scorecard, limit and affordability calculations, and post-Bureau rules in order to obtain the customer's score, final approval, and limit offered. These results are updated on SAD. The results are finally displayed and the customer is informed of the decision where after the fulfilment process takes place.

In the case of a valid record being available on BAD, the SAD application is updated and the same subsequent process is followed.

4.3.2 The software development model – current state analysis

This section describes the software SDLC of the credit scoring process in terms of the software development model it is designed upon:

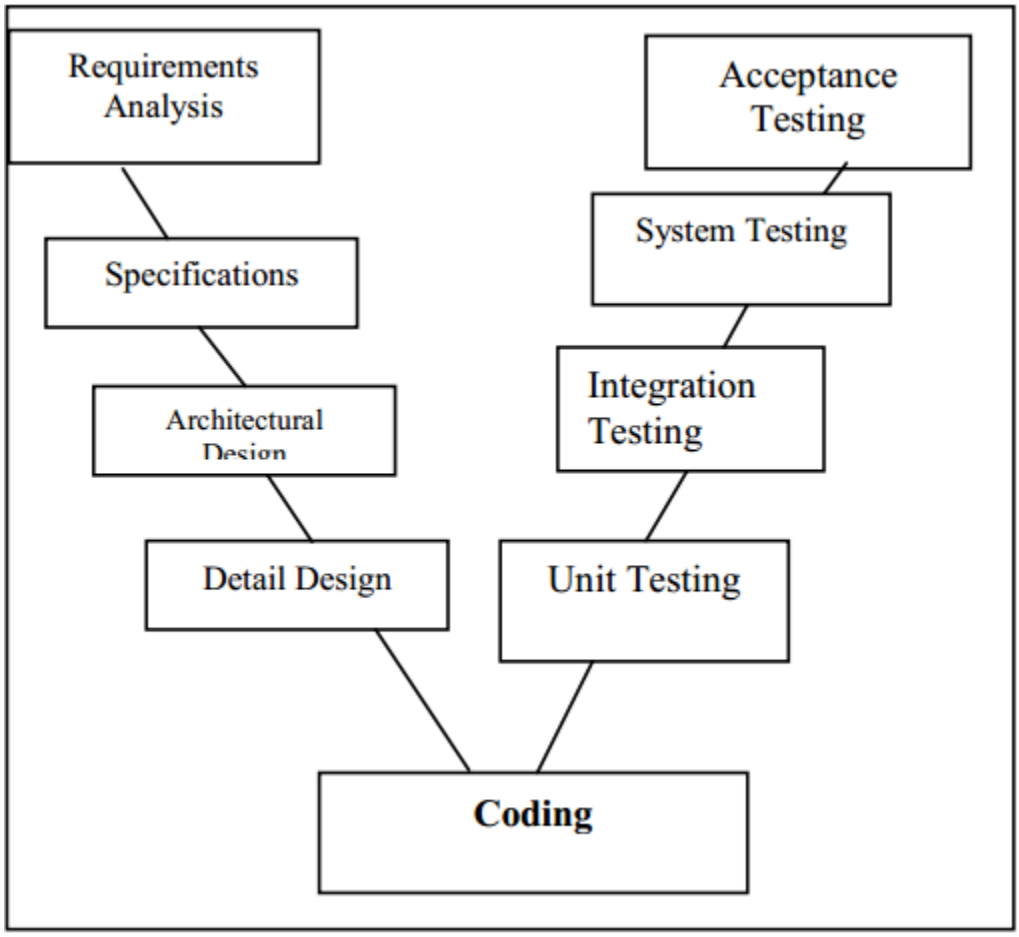


Figure 4.3 The V-model (Mathur & Malik, 2010)

The V-development model

It often occurs that changes need to be made to the current credit scoring logic, which includes the credit origination scorecard, limit and affordability calculations, pre-bureau rules and post-bureau rules. These changes could include new a new scorecard, calculations, rules, or amendments to the current scorecard, calculations, or rules.

When these changes are requested, a project is logged and initiated in order to develop a new software system that contains the software changes. The current software development process that is involved in making a software change is developed according to a traditional V-model design as depicted in Figure 4.3.

The V-model has a corresponding test activity for every development activity. Analysis and design of tests begin during the corresponding development activity for each level. The design of the V-model allows for risk management.

There are various risk factors involved in testing such as risks relating to schedule, requirements, human resources, and quality. Risks in schedule are related to unrealistic project schedules requested by the relevant business unit. Therefore expectations are managed during the requirements analysis phase in order to avoid this as far as possible. The project could face several risks due to user requirements. This could be due to lack of clarity in user requirements, ambiguous requirement definitions, changes in the requirements or unrealistic requirements.

Testers are involved in reviewing the specifications as soon as possible in the lifecycle, in order to pick up defects in the specifications early on. Factors such as unrealistic schedules, lack of resources, and frequent requirement changes could compound the risk of poor quality of the system. Therefore not only are business expectations managed as far as possible, but requirements analysis is also performed in detail in order to develop an optimal and realistic requirements definition. The project could face risks if there is a lack of human resources available with the necessary skills required in the project. These issues are therefore addressed during impact analysis, project sizing and scheduling meetings.

Quality is another risk factor of a project. Therefore testing is conducted at all levels of software development in order to ensure that defects are not present at a particular level and then later built into larger parts of the software system.

Testing the developed software is required in order to ensure that the level of quality is on par with the specifications, to provide information for decision making, to prevent defects, and to mitigate risk. Testing involves analysing the tests, designing test cases, implementing the tests, executing the tests and finally, comparing the test results. Various types of testing are performed at all stages of the V-model to test for changes that were made to the software.

The test analysis and design begins early in the development process followed by testers reviewing the development documentation. Testing is already performed early in the lifecycle in order to ensure optimal quality of the process. This approach to testing is costly and time-consuming but ensures that the correct results are achieved by the end of the project.

If tests are designed as early as possible, defects in the specifications will be found early in the process when they are still inexpensive to fix. If the testing is done too late in the lifecycle, time and effort may well be saved but testing quality could be compromised as defects could be found much later when they are more expensive to fix. In addition, the defects in the higher levels, such as the requirements specification, will be found late. These defects are the most critical and important as defects in the higher levels will be built into the lower levels of the V-model.

Another reason the tests are designed early according to the V-model is to ensure that the order in which the system should be put together for testing is defined during the design phase. The order of software development is specified before it is built. Testers are thus not constrained by the order in which software is built as tests are designed early in the lifecycle according to the specifications and do not rely on the software to be built first in order to initiate testing. This allows the integration of the system to be known early therefore the system can be built based on the known integration. This can greatly reduce development time later on as development activities and verification of these activities are performed early.

Testing time is reduced as testing activities are broken down to be performed at different stages of the life cycle. For example, the first integration can be built and tested before continuing to integrate more parts. This also allows integration testing to be performed concurrently with development of the system, thus saving time but not necessarily effort. When problems are picked up, they are immediately fixed before proceeding. If testing is only done once at the end of the process when the entire system has been put together, it might be difficult to find the cause of the defects as the cause could be at any of the levels in the V-model.

In summary, the benefit of designing tests as early as possible is that quality is built in, costs are reduced, and time is saved as fewer defects are found. Therefore, quality is built into the software development process.

Testing is performed according to specifications, which dictate what the correct results of the testing should be. Specifications are documented in order to ensure that there is no misunderstanding or lack of clarity. Specifications are designed by taking testing into consideration. Knowing how the tests will be performed assists in developing the specifications for development. The specifications therefore simplify the process of setting up test cases. Testing reveals defects in code, parts of the system, the system as a whole, or the user's view of the system. Specifications are compiled in all levels of the V-model, from a business requirement specification to a specification for the code. The tester, quantitative analyst or developer performs a test comparison that detects the differences between the actual test results and the expected test results by using the business requirement specifications. The business requirements are validated by discussing them with users in the business unit and comparing them against the knowledge of the business unit's working practices.

If a defect is detected at any stage of testing, the defect is reported and subsequently resolved by recoding the software. A new version of the software is released containing the fixed defect. Before being able to continue to the next phase of testing, re-testing is performed on the latest software version in order to ensure that the defect has been fixed correctly and the rest of the system is still working as per the requirements. Re-testing is executed in the same environment and using the same test cases. The reason that all of the test cases are executed again and not just the test case pertaining to the defect is that the fixed defect could possibly introduce new unexpected defects elsewhere in the system. This might however still not be picked up as only one part of the system has been tested. Regression testing is finally performed in order to address this issue.

Regression testing tests whether the changes made to a particular part of the software has not caused secondary problems elsewhere in the software system. This ensures that modifications in the software or environment still meet the original requirements without causing unintended side effects in the system. It is performed every time changes to the software or environment are made. Regression testing is performed at all levels in the V-model in order to ensure that the requirements are met at all levels. In order to simplify the process, regression testing is

automated through the use of mathematical algorithms that are applied to a set of test cases. The test cases currently consist of only one customer profile with account data.

Component development and testing

The lowest level in the V-model is component development and testing. A component is a small piece of software that is one of the building blocks of the software. A piece of software could be a few lines of code, a small program or several database modules. The component is thus the lowest level item that is testable and is tested in isolation if possible in order to ensure that it is tested in detail. The purpose of component testing is thus to test the detail of the coded software.

Once the software component has been coded by the developer, the component's functionality, structure and interfaces are tested in the development environment. If defects are detected, they are resolved as soon as they are found by recoding the software. The defects are recorded as this might assist with detecting and solving defects in other parts of the system.

Once the component has been coded by the developer, component testing is performed by the quantitative analyst and the developer. The developer is able to find defects and their causes quickly as he or she understands the logical flow of the code.

If component testing were to be performed by someone other than the developer, this could result in more time spent on testing as it would take another resource much longer to find the cause of the defects and each defect would have to be documented, reported and explained to the developer. The developer would then have to review and analyse the reports and fix them. Before component testing is performed, the tests are designed by the developer based on the component specifications and code. Both functional and structural test cases are designed depending on the risks, importance and complexity involved.

Integration development testing

The next level of the V-model is integration development and testing. Integration development and testing is performed by combining components that have already been tested into larger assemblies so that testing can be performed on the newly formed assembly. Testing on the

assembled part is performed by testing that the components correctly function together by looking at the interfaces between them. This assists in detecting defects that were not previously detected when the components were tested in isolation. It can occur that when components are combined, certain aspects of one component could result in functional failure of another component. For example, although interfaces were tested during component testing in order to ensure that communication occurred from one side, integration testing could pick up that the communication between the different interfaces is not working from all sides.

The objective of integration testing is thus to test the interactions of the integrated software part. Integration testing is based on the software system design, architecture, and test cases. During integration testing, not only is the functionality of the interfaces tested, but non-functional quality characteristics are also tested such as performance and structure. It is important to pick up performance and structural issues earlier in the process as performance and structural changes may occur as more and more parts are assembled. Resources are planned before integration testing commences.

Integration development and testing is performed at multiple levels of the V-model, such as at component development and integration testing and system development and integration testing. Component integration testing involves testing the interfaces and interactions between components that have been combined. It is performed just after component testing has been conducted by the Hogan technology quality assurance (HTQA) testing team. System integration testing involves testing the interfaces and interactions between smaller parts of the system, including hardware and software, that have been combined to form the final bigger meta-system. This is performed just after sub-system testing by the HTQA testing team.

Integrating systems and components can be a complex task when many different components, systems, interfaces and areas of the organisation (business units) are involved. Therefore, integration is done in increments or steps based on the architectural design of the system. Components are combined into an assembled part and tested. This ensures that the basic parts are integrated and tested first before continuing to more complex integration. This is followed by combining the different tested assembled parts and testing it as a new assembled part. This continues until the entire system has been integrated and tested.

Thus a bigger part of the system is assembled only when the smaller parts have proved to be working and are trusted. This process is known as incremental integration. Incremental

integration is performed in order to discover and fix defects easily and at the earliest opportunity but also offers faster and easier recovery if defects are detected.

In order to ensure that integration testing is performed as efficiently as possible, the testers of the HTQA team are involved as early as possible in the development of the software. Before integration testing commences, the test analyst determines the integration strategy by determining how many components need to combined in each step and the sequence of combining the components. This is done by looking at the architecture, functional tasks, and processing sequence.

The test analysts of the HTQA team prepare test cases according to the test strategy, prepare the test data, test according to the requirements and design, compare test results and manage defects. The testing team has a thorough understanding of the software architecture, which assists them in performing integration testing. The testers first plan the order in which the tests will be integrated. The order in which the actual components or system is integrated is therefore based on the order of integration testing. This results in more efficient testing as well as reduced time spent on development.

System development and testing

System development and testing is performed at all stages of the V-model, which combines and tests the system as a whole. System testing is performed by the HTQA testers on the entire system in a realistic environment. It involves functional, non-functional and structural testing. Tests are based on system and functional specifications.

Firstly, functional testing is carried out in order to ensure that the system is functioning according to the specifications. Non-functional and structural tests are subsequently performed on the system to ensure testing thoroughness. The environment in which system testing is carried out is realistic and representative of the production environment in order to ensure that the same factors are taken into account and therefore environmental defects are avoided.

Functional testing assesses the behaviour of the software in order to determine whether it is functioning according to the requirements detailed in the functional specification that specifies exactly what the system does and how. Test design is based upon these requirements. The

requirements as well as the tests are subsequently prioritised based on risk criteria. This ensures that the most important and most critical tests are performed.

Non-functional testing assesses how the system is performing according to the schedule. Quantifiable units of measurement are defined in order to determine exactly how well the system is performing. Testing is performed according to requirements specifications and is performed in a realistic environment that resembles the production environment. Examples of non-functional testing performed include performance testing, load testing, stress testing, maintainability testing, reliability testing, portability testing and usability testing.

Performance testing measures the performance of the software by performing timing tests such as response times. It monitors the behaviour of the system by logging the number of transactions and response times of these transactions. Reports are produced based on the logs and graphs of these load versus response times. Performance testing tools, such as a user interface or test harness, are used to generate realistic loads on the system, database, or environment in order to determine the behaviour of the system.

Load testing is performed by testing a volume of records that is representative of the production environment in order to verify that the system will be able to handle the volume of records in the actual production environment. During load testing, aspects such as processing throughput and the number of connected terminals are tested.

Storage testing is performed by testing whether the objectives for storing are being met. There can be various objectives for storage. For example, a limited amount of space on the server may be used for certain operations or a limited amount of customer data may be kept in the data warehouse for certain customer profiles. Storage testing is usually applicable to embedded software, for example software embedded into the various data warehouses.

Portability and interoperability testing is performed in order to ensure that the system will be able to operate in the different configurations. Testing is performed in a representative set of configurations of the production environment. Configurations refer to the versions of the software. For example, configurations used by the business unit (user) could be the different versions of the credit scoring system. One version of the credit scoring system could be used for monitoring and another version could be used for applications.

Portability and interoperability testing also involves testing the interconnected data paths in the system. This is especially important when a part of the system has been upgraded as this can

now result in conflict with other parts of the system that have not been changed. Lastly, portability and interoperability testing is performed in the final production environment as the production environment may have physical characteristics that can influence the behaviour of the system. The same system could behave differently in different environments due to the different physical attributes of the different environments.

Reliability testing is performed in order to determine how reliable the system is. Reliability refers to whether the system has a low probability of failure. Stress testing methods are used to conduct reliability testing. Stress testing is performed by testing whether a volume of records beyond those found in the production environment can be handled by the system. This serves as a safety and risk measure to ensure that the system will be able to handle additional number of records added to the system in a production environment.

Usability testing is performed by the business unit (users) in order to determine how usable the system is for them. The reason for testing is performed by the users instead of the technical staff is that it is often difficult for people from a strong technical background to know whether the users will experience the system as being user-friendly as they might perceive a more complex system as quite simplistic due to their computer-literate skills.

Qualities such as reliability, maintainability, portability, and availability are defined in measurable terms in order for testing to be performed. If these qualities are not defined in measurable terms it will be difficult to analyse the results of the tests. This is because it would be prone to subjective testing that is not reliable as it could differ from person to person. An example of a quantifiable or measurable quality is the time taken to make a change.

Once the system has been moved to the production environment, maintenance testing is initiated in order to ensure that the system remains functioning as per the requirements and that the quality of the system is maintained. Maintenance testing is also performed when changes to the existing system in the production environment are made.

Aspects of sample changes made to the system are tested. These changes can be due to modifications, migration to new platforms or retirement of the system. Examples of modifications to the existing system include fixing of defects and enhancements and upgrades of the system such as changing the infrastructure or upgrades to the software. Modifications could also include corrective and emergency fixes and environment changes.

Maintenance testing is performed by the testers of the HTQA team. The testing of changes is in depth and focuses on the specific areas where the changes have occurred. It is performed by testing the entire integrated system first in order to see the effect on the whole system. Once the results are satisfactory, the specific area where the change was made is then tested.

Production testing is easier than testing during development as live data is available and it is therefore not necessary to go through the process of building test data. Maintenance testing also involves performing regression testing. Regression testing needs to be performed as the software changes made can cause other areas of the system to be affected. Regular testing of changes may not pick up all possible defects as this only involves testing the area of the software pertaining to the change. Once the system has gone live (i.e. moved to a production environment), monitoring is set up to observe the system going forward for any defects. This information is continuously analysed and verified to assist with the traceability of defects.

Structural testing is performed by testing the architecture of the system or component in order to determine how thorough the testing up to a certain point has been. The architecture refers to the process logic of the credit scoring software. This testing complements functional testing by testing a set of conditions that cover different elements of the architecture. It provides a measure of how thorough the tests have been in order to establish if more tests are needed to gain the necessary coverage of test cases. Structural testing is performed at all levels of the V-model where it is deemed necessary. For example, it is performed at component level in order to test the code coverage, integration level in order to test the module coverage, system level in order to test the final system coverage, and finally at acceptance level to test the business model coverage.

Acceptance testing

After system testing has been completed, acceptance testing is performed. This is the final stage of validation. User acceptance testing is mainly performed at the end of the development process when the entire system has been integrated. Testing is performed by the final users of the system in order for them to gain confidence in the system and to determine whether the system is ready to be transferred to the production environment. It is therefore the responsibility of the users to ensure that the system is working according to their initial requirements.

The business unit is involved in developing the specification for acceptance testing. This is done at the beginning of the project, thus at the beginning of the system development. The business unit is involved in reviews throughout the project and in the design of integration system testing. This ensures that the final product (i.e. system) is built correctly according to the business specifications and ensures that the business unit understands the system and it's complexities in detail.

Acceptance testing is performed by executing any test cases that the users deem necessary in a different test environment. Technically, not many defects should be present at this phase of testing as most of the defects should have been resolved during functional and regression testing. Acceptance testing occurs at various levels of the V-model. Acceptance testing is performed after component testing to ensure that the components are usable and it is performed after system testing or system integration testing to ensure that the functionality of the system is aligned with the user specifications. Typical defects found during acceptance testing are mismatches with business needs and misunderstandings of business processes. After user acceptance testing has been performed and the business unit has found that the system validates, sign-off is given by the business unit in the form of a mail stating that the results proved to be satisfactory.

4.3.3 The software systems development life cycle – current state analysis

The process of the software SDLC pertaining to the credit scoring process in the financial institution is dealt with in this section.

In the event of a change requested to the rules, scorecard, or affordability and limit calculations, software is developed in order to make the relevant change. This also includes cases where a new scorecard has been developed and requires implementation. The software development process is made up of a development life cycle, which contains the different phases of development as depicted in Figure 4.4, which presents the researcher's schematic illustration. Figure 4.5 details the SDLC at the case site. Figure 4.6 provides a high level overview of the entire software development procedure that is initiated when a change is requested to the software. This provides a high level explanation of the software SDLC.

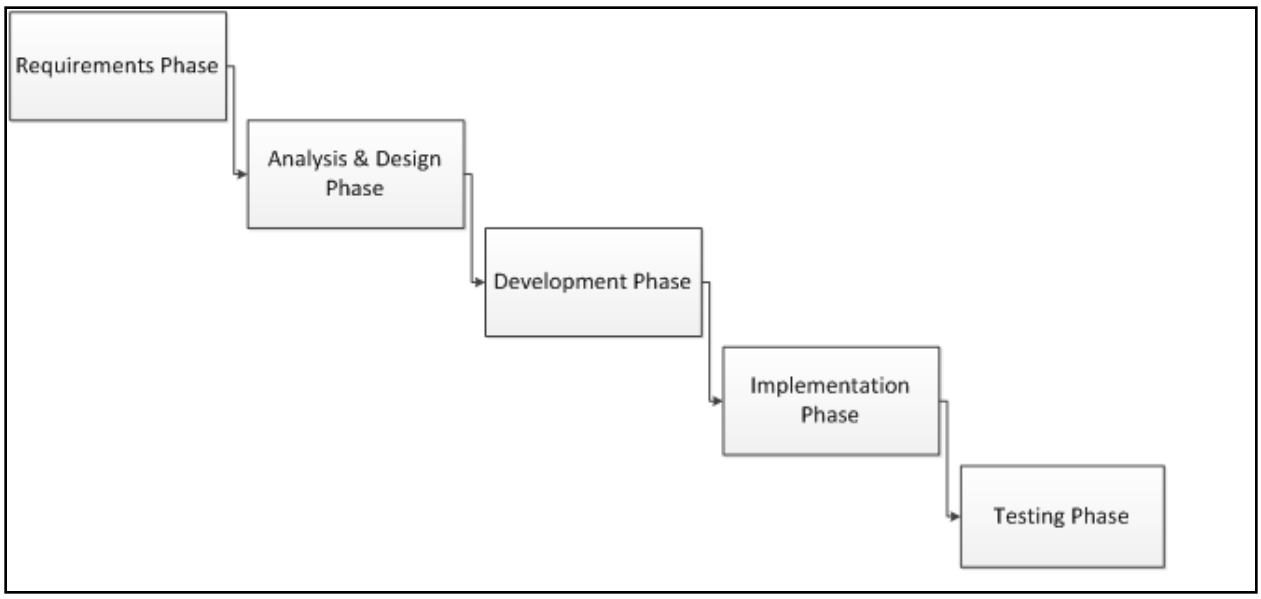


Figure 4.4 The Software SDLC Phases

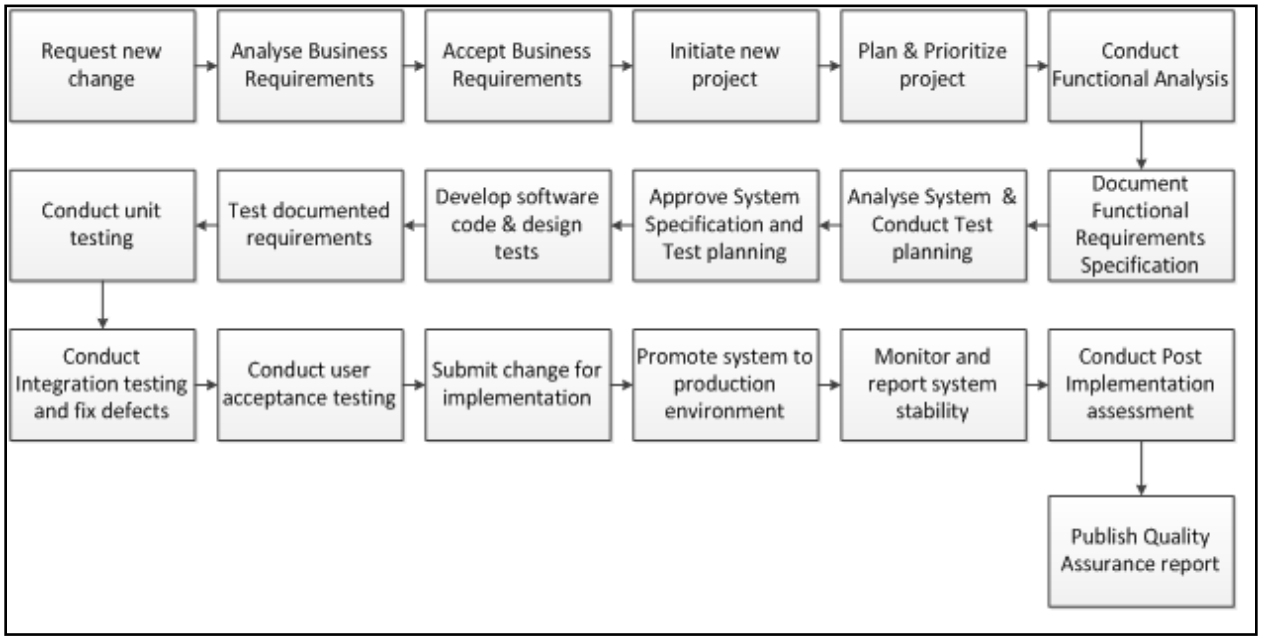


Figure 4.5 The Software SDLC Phases in Detail

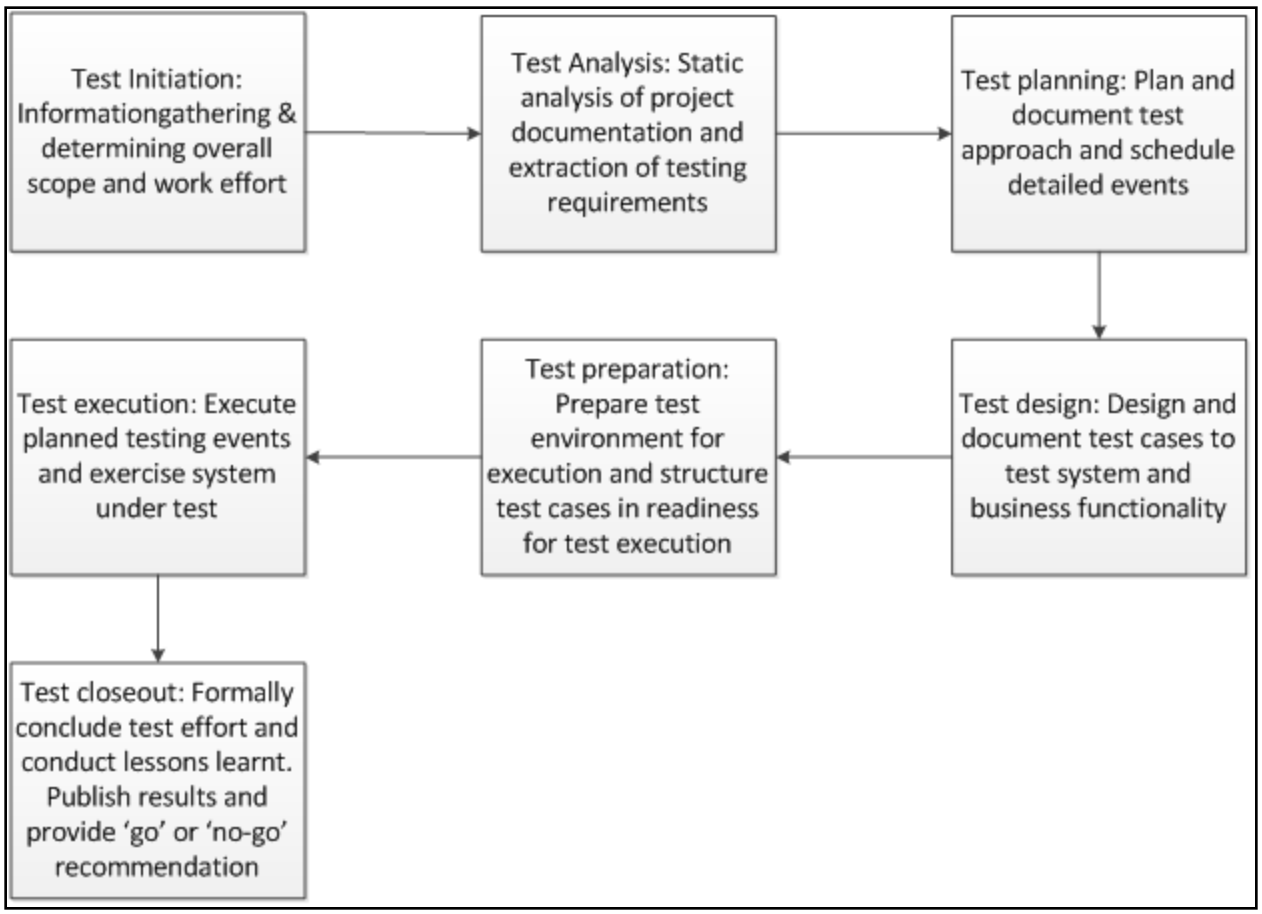


Figure 4.6 Overview of the Software Development Procedure

Requirements analysis phase

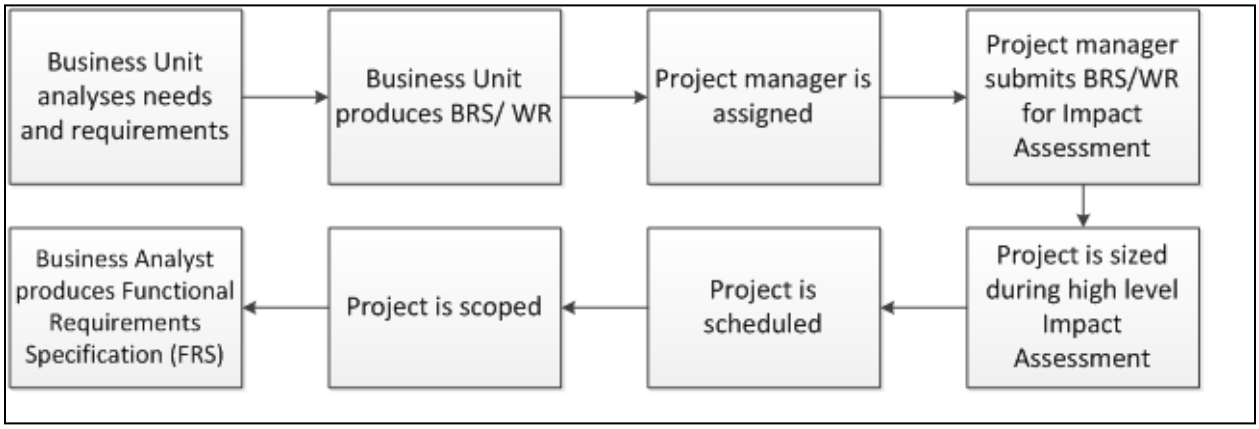


Figure 4.7 The Requirements Analysis Phase

The requirements analysis phase as represented in Figure 4.7 is where the requirements of the business are determined and analysed. The business unit defines and analyses their own needs and transforms it into a requirements specification. Once the business requirements are developed by the relevant business unit, it is submitted to the analytics team in the form a business requirement specification (BRS) document. This allows the business unit to formally request the changes. The BRS contains specifications of the development required.

The requirements are logged as a project and a project manager and project team is assigned. The project team consists of the resources from the various disciplines involved in the SDLC such as the analysts, developers, project manager, divisional managers, and testers. The BRS is reviewed by the project team in order to determine if the information provided is sufficient for development. The project team also determines whether the requirements (functional and non-functional) are realistic and whether it will achieve the desired objective of the business unit (stakeholders).

Communication with the business unit requesting the change is initiated in order to confirm the details of the requirements and understand and define the problems and objectives. Should the information provided be insufficient, the BRS is sent back to the business unit for amendments to be made. Afterwards the project team will once again review the specification.

Once the project team finds the information provided to be sufficient, it is submitted for a high level impact assessment. The project team, project manager and the business unit (end users) are involved in the high level impact assessment. During the high level impact assessment the project is sized, dependencies and impact on other areas are determined, and additional information is requested if required. Standard language is defined in order to ensure that all the resources and stakeholders have a mutual understanding. Factors such as technical size, technical complexity, technical quality, technical value, project duration, project cost and risk are taken into account and discussed. The current environment is assessed and performance, operational, functional, and interface requirements are formulated, quantified, and refined according to the business' needs.

All possible solutions are considered so that the optimal solution can be reached. The best requirements and solution are selected by comparing them to the business' objectives. It is ensured that the requirements are clear, unambiguous, accurate and complete. The requirements include details on how the problem should be solved (how the system should be

designed), how well the problem should be solved and how the system should be verified and validated.

Security requirements are developed in order to conform to the Information Security Standards (ISS). Design constraints are defined and clarified. The resources from the various disciplines and divisions in the SDLC as well as end users provide comment, raise concerns, and give their opinions on how to proceed. The outcome of these meetings is that all involved understand the entire system, the trade-offs, the derived problem definition, and that agreement is obtained so that everyone including the stakeholders will accept the solution.

Risk is considered during the selection of users' objectives and design. Knowledge regarding the technical aspects of the system is shared between the project team resources and end users. Successes and similarities of previous projects and lessons learnt by the different resources are discussed and taken into account in order to reduce the probability of defects and improve quality and schedule. The implications of changes made, the external and internal influences on the system, and the probability of successful completion of the project within budget and schedule are determined. Risks are identified, listed, quantified, analysed and prioritised. The probability of risk occurrence, seriousness of impact, assessment of impact on cost, schedule, and performance, sensitivity to assumptions, and degree of correlation amongst risks are determined.

Alternatives to mitigate risks are defined and evaluated. Preventative and contingency measures are set in place. Risks are tracked throughout the SDLC to ensure that mitigation plans are effective and the potential impact on the project does not increase.

Agreement is obtained on timelines and funding required. Commitment by all involved is obtained. Finally, the requirements are prioritised and categorised in order to ensure that the critical requirements are met first should all the requirements not be able to be completed within schedule. The requirements are subsequently organised from high level to low level requirements in an understandable and traceable manner. Finally, the functional and non-functional requirements are allocated to the relevant system elements.

After the impact assessment, the project is scheduled. During scheduling aspects such as the business priority, areas involved, dependencies on other projects, and whether a functional requirement specification (FRS) is required are determined by the project manager and project team members. The FRS specifies the changes required on a functional level and contains the

integrate picture of the system. The FRS is developed by identifying and decomposing the system's functions and is detailed enough in order to construct the system accurately. The system functional architecture is designed in a logical and orderly manner according to customer needs and technical capabilities. Development activities are planned, a preliminary system design is developed and the development activities are translated into requirements. Requirements are developed for each function.

After the project has been scheduled, the project scope is confirmed. The project manager, project team members and business unit is involved in the scoping session. The business unit presents the project outline and gives an overview of the changes and requirements involved. The project is once again examined in order to determine if anything has changed. If changes have been made, the process is repeated from the impact assessment phase. Otherwise, the requirements are finalised and whether a FRS is required is finally determined. Should it be required, the business analyst develops the FRS. The project manager estimates the time, cost, scope and resources according to the specifications and risk.

Throughout the SDLC, the project manager is responsible for planning the project according to the desired timelines and budget, developing a work breakdown for all resources, ensuring that the duties and responsibilities of all involved are executed according to the schedule, ensuring that the different elements, components and interfaces are combined and tested in an efficient way in order to meet the stakeholders' requirements, and keeping track of and documenting requirements changes. The project manager is therefore responsible for having a thorough understanding of the SDLC, its complexities, weaknesses, strengths, interdependencies, processes, and the environment in which the system will operate, and assigning resources accordingly. The stakeholders are kept involved throughout the entire process in order to ensure that the system is being developed according to their needs. The identification, mitigation and resolution of risks proceed throughout the SDLC by the project manager.

This phase is validated and verified by analysing and reviewing the requirements and by comparing them to the business unit objectives. Validation is also done by documenting the requirements definition, analysing the cost and benefits, as well as reviewing the preliminary project plan, risks and contingency plans.

The outputs of this phase are the approved project scope and objectives, and initial estimation of cost and benefit, project priority, a preliminary project plan, a tentative release date, approved

business requirements, assessment of all technology involved, and a project life cycle assignment.

The deliverables are the BRS and FRS. Concerns regarding this phase are the changing of requirements by users during later phases in the SDLC and inadequate information regarding previous similar projects.

Analysis and design phase

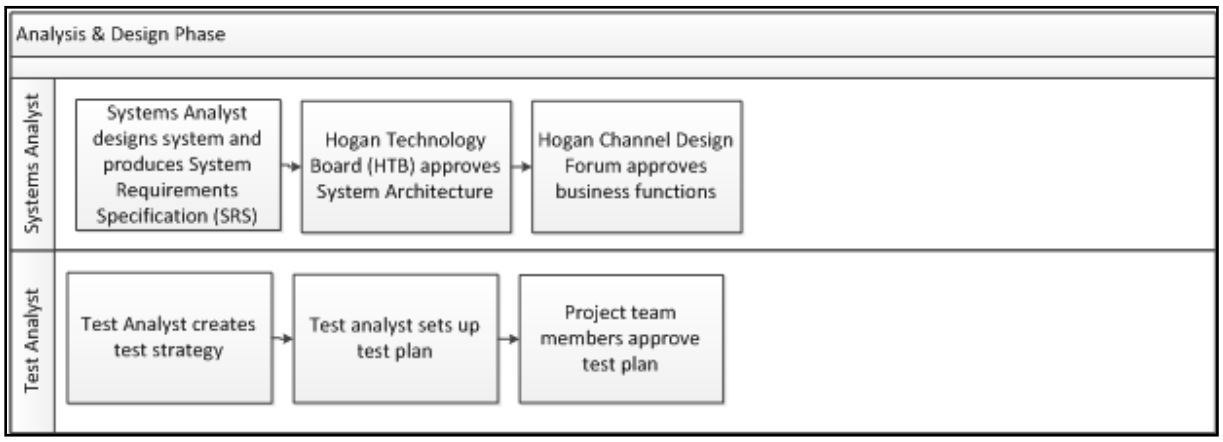


Figure 4.8 The Analysis and Design Phase

The analysis and design phase is initiated once the outputs of the requirements phase have been concluded. The analysis and design phase is represented in the researcher's illustration depicted in Figure 4.8. This is where detailed requirements are transformed into a complete detailed design of the system by focusing on how to deliver the required functionality.

The systems analyst designs the system according to the system functional architecture as detailed in the BRS and FRS. The design involves detailing the function and performance of the system, developing the required architecture and defining how the system should be verified. The various elements and subsystems are defined. The design includes the interfaces between different elements, subsystems and systems. For example, the design includes investigating how to send the data to SM and other business units as well as investigating which information

is required from the various business units in order to develop the system as per the requirements. The system is analysed by identifying all interacting components and systems involved, considering all possible solutions that would address the problem and determining the system boundary. Alternative design solutions are evaluated based on aspects such as performance, schedule, cost and risk. The system is therefore designed by considering the entire system life cycle.

The most optimal design solution is selected by developing different models and prototypes of the different solutions and running trials in order to identify the solution that produces the best outcome. It is ensured that the design is simple and understandable by the developers and end users. Input and review from other resources and the end users are obtained. The systems analyst produces an SRS that specifies the changes required to the current system. The SRS specifies where in the system and how the functional changes are going to occur. It is ensured that the SRS is detailed, clear and accurate. The technical design is verified and validated by ensuring that the functional architecture can perform the required functions and required level of performance according to the BRS and FRS.

The resultant designed architecture is approved by the Hogan Technical Board (HTB). Subsequently, the Hogan Channel Design Forum (CDF) approves the required changes to the business functions.

Parallel to this, the test analyst proceeds to develop the test requirements and design the test strategy. The test analyst analyses the FRS and documents the testing requirements in a test plan document by drafting the process of the test roll-out with start and end dates for each test case.

The project plan and system test plans are reviewed by the project team members and project manager. The outputs of this phase are a detailed technical design detailed in the SRS, a test plan and project status review. The project status review involves assessing the status of the project in terms of timelines, objectives achieved, cost incurred in relation to budget, and a review of risks. This phase takes approximately four to eight weeks to complete. Available resources during this phase are the project manager, project team members and end users. A concern of this phase is inadequate information available on previous project designs.

Development phase

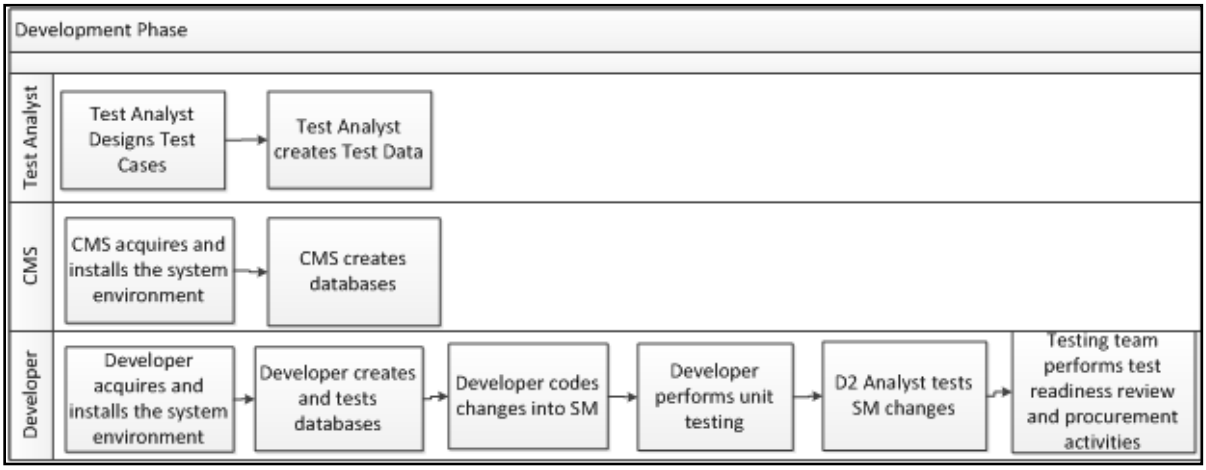


Figure 4.9 The Development Phase

During the development phase, which is depicted in the researcher's illustration in Figure 4.9**,** the design is converted into a complete information system. The test analyst designs the test cases to match the BRS specification. This involves creating a set of conditions or scenarios under which the tester determines whether a project is working according to the user needs.

The test analyst then tracks the test coverage by means of a documented test coverage matrix (TCM). A TCM is constructed by creating a checklist which ensures that the functionality of each software unit is checked in all possible combinations (positive as well as negative, and special conditions). The outcome is in the form of a matrix document. This is done to ensure that all probable conditions and cases for a feature to be tested are thought through. It also assists in identifying probable gaps. The test cases and scripts are reviewed, modified and refined by the testing team.

The test analyst proceeds to create the test data. This involves setting up the variables and fields necessary for testing. The test data is manually manufactured by the test analyst by developing scenarios that test the software robustness. These scenarios cover all possible combinations related to the software change: once the testers have determined which scenarios need to be tested, they create a customer record that relates to each scenario. Approximately 20 to 30 customer profiles are manufactured. Parallel to creating test cases, the CMS and the

developer acquire and install their respective system environments (e.g. integration testing or INT, and the quality assurance or QA) and proceed to create and test the databases. This usually involves changing and adding segments to the existing databases as well as refining programs. The test plan and scripts as well as the project plan is reviewed.

System development occurs once the outputs of the design phase are concluded. The developer performs integration by coding the software changes into SM according to the SRS, FRS, test plans and coding and infrastructure standards. The developer and quantitative analyst subsequently proceed to perform unit testing (component testing). During unit testing, the developer tests whether the coded input is resulting in output, but does not test whether the results are correct.

Once this has been done, approval is given by the developer that the coded changes in SM have been completed. The quantitative analyst proceeds to verify the code by testing whether the coded changes made to SM are resulting in the correct output. This is done by developing test cases that test the different scenarios related to the change. Mathematical algorithms are used in order to automate unit testing. These algorithms run the test cases through the data obtained from the production environment. Once the code has been verified, the quantitative analyst provides sign-off confirming that unit testing has been completed. Unit testing code is stored in order to be re-used during post-implementation testing or another change request.

The developer integrates the system incrementally according to the technical design and ensures that each integration step achieves the desired system functionality. The developer ensures that the system is integrated and interfaced with other systems as required, that all operational systems are compatible with each other, and that the correct versions of and correct components are used for integration.

Finally, the testing team performs a test readiness review in order to verify whether everything at this stage is ready for the testing phase to commence. The scope, risks and issues are analysed. Procurement activities are performed to ensure that the appropriate resources are allocated to the phases that follow. The project plan, code and infrastructure changes, and unit test results are reviewed by the project manager. This phase takes approximately two to six weeks to complete. The available resources are the test analyst, CMS, the developer, the D2 analyst and the testing team. The deliverables are the coded solution into SM software, the test pack, the test plan, code scripts, unit test results, code and infrastructure reviews, review of

project status, scope, risks and issues. A concern of this phase poor verification due to limited test data created.

Testing phase

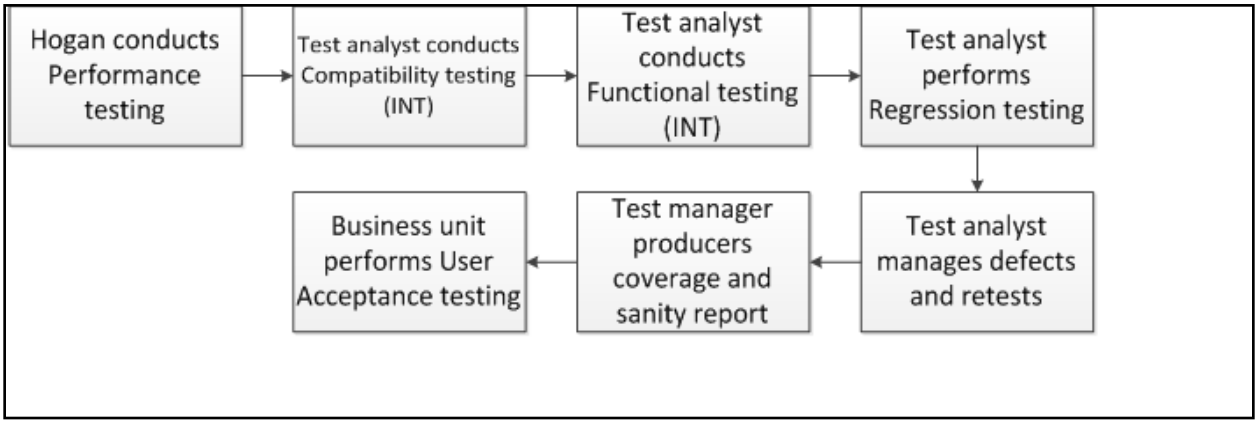


Figure 4.10 The Testing Phase

The testing phase commences once the outputs of the development phase have concluded. During the testing phase as depicted in the researcher's flow chart depicted in Figure 4.10, Hogan conducts performance or load testing. A large number of records, which exceeds the number of records in a production environment, are executed through the newly developed software. This is conducted in order to verify that the system will be able to handle a realistic amount of records in the production environment. It also assists in verifying that the integrity of the data is not comprised. Reports are produced and analysed.

Compatibility testing is performed. This tests for compatibility with other channels, operating systems, old or new versions and target environments. The outcome is presented in the form of a test case document.

Compatibility testing is followed by integration testing. Integration testing consists of two separate phases: the integration testing phase (INT) and the quality assurance testing phase (QA).

Functional testing is performed during the INT phase on the integrated components and system by the test analysts. Functional testing is performed in order to demonstrate that the developed system conforms to the requirements as specified in the functional requirements document. It is performed by conducting component integration testing, system integration testing, and finally system testing. It involves both positive and negative testing.

The test analysts proceed to manually manipulate the relevant input of each manufactured customer record to see if the desired output is obtained. For example, a requirement is to decline all customers for a loan that have a monthly net income of less than R 10 000. The test analyst extracts a customer record where the customer has a net monthly income of less than R 10 000 to verify that the application is declined (positive testing) and a customer record where the customer has a net monthly income of more than R 10 000 to verify that the application is not declined (negative testing). The test analyst thus verifies that the customer is approved or declined correctly. The outcome is in the form of screen shots. Test results are documented and archived for future use.

Structural testing is performed against the SRS by the test analysts to complement functional testing. This ensures that testing has been performed thoroughly by testing different scenarios that pass through the different flows in the architectural design. For example, a customer who has an income greater than R 10 000 per month might pass through different scoring logic compared to a customer who has an income less than R 10 000 per month. Test analysis reports are produced during this stage.

Should defects be discovered during functional testing, the test analyst will resolve the defects with the developer and retest the defects in order to ensure that the problem has been resolved. The test manager will proceed to produce a defect and test coverage report in order to verify that possible scenarios were not left out. The changes are subsequently approved in order to migrate the system to the QA phase. The deliverables are the test case execution and functional testing. The available resources are the project manager, technical team lead, developer, business analyst and test analyst. The INT phase takes approximately four weeks to complete.

QA testing follows functional testing. During QA testing, the test analyst performs regression testing. In the current SDLC, a regression test pack is used that consists of one customer profile that is manipulated according to the test cases that cover the change in order to verify that the correct output is obtained. Positive testing is performed on the customer profile. For example, if

the change was to approve customers with a monthly net income of greater than R 10 000, a customer profile is selected from the production environment that has a monthly net income greater than R 10 000 to verify that the customer is approved. However, tests are not conducted to verify that customers with a net monthly income less than R 10 000 are approved. Automated regression is performed that use test execution tools in order to execute the tests. The outcome is in the form of a test results document and screen shots.

The available resources are the project manager, the technical team lead, the developer, the business analyst and the test analyst. The QA phase takes two weeks to complete. If any defects are picked up during regression testing, the defects are resolved with the developer. The test analyst compiles a defects list which is a matrix of defects, showing the description of the defect, the date the defect was identified and the person the defect is assigned to. The test analyst finally retests the defects until the results are satisfactory.

Once testing has obtained the desired results, the test manager produces a test coverage and sanity report of the cycle. The testing team subsequently approves the migration of the system into the production environment. Available resources during this stage are the project team members, project manager and business unit. The deliverables are the test coverage report, defect report and project status review.

In order to validate the system, acceptance testing is performed by the relevant business unit with the help of the development team. The objective of user acceptance testing is to allow the business unit to gain confidence in any aspect of the system, determine whether the system meets their needs, and determine whether the system is ready to implement. The business unit tests a random selection of data in order to determine whether the business requirements are met. The business unit uses any test environment and performs any test they deem necessary.

The business unit designs their tests at the same time as developing the BRS. This helps the business unit to ensure that their requirements are logical and realistic. Otherwise, the business unit may realise only at the end of the development process that their requirements are incorrect. This will result in requesting changes late in the SDLC that can cause the schedule, cost, and quality to be compromised.

Once the business unit has accepted the system, the project status is reviewed and formal approval is given to implement the system into the production environment. The outputs of this phase are the test results, the users' approval to implement the system, handoff of the

operational technology, and reviews of project status. Concerns of this phase are long timelines, high costs and poor confidence in verification results.

Implementation and maintenance phase

The implementation phase as depicted in the researcher's flow chart in Figure 4.11 includes the preparation and implementation of the system into a production environment. Once business unit acceptance is received and positive test results are obtained for system and regression testing, Hogan submits the change for implementation by producing the final package to install into the live (production) system. The data is then migrated from the QA environment to the production environment according to the configuration management details. Subsequently, the analytics team and business unit extract a sample of records to verify whether the data is flowing through the correct path of logic.

Finally, the system is implemented into a production environment. The system is operated according to standard system operating and management procedures. Once the system is in production, the test analysts performs maintenance testing which is known as post-implementation testing (PIT) in order to verify that the environment is stable and that nothing is broken (i.e. that defaults do not appear in the production environment).

Data from the production environment is extracted and the data is run through the same logic used during unit testing. The system is monitored in order to make changes where necessary and reports on the stability of the system are produced. Depending on the scope of the changes required, previous life cycle phases are revisited where necessary. A QA report is subsequently produced in order to state that the system is stable and that no defects are occurring. Site acceptance testing is performed by the business unit to ensure that the system is still validating in the production environment.

Maintenance of the system is initiated once the system has been moved to the production environment, the system has been verified and validated, and the project has been signed off. Maintenance involves assigning resources to ensure that the different components and system as a whole is functioning. Maintenance testing is conducted on a daily basis in order to ensure that the system is still functioning according to the requirements.

Improvements and adjustments are continually made where necessary in order to ensure that the system keeps on working as required. The available resources are the project team members, project manager and business unit. The deliverables are the post-implementation review, the configuration management records and the project closure. Concerns of this phase are that changes are often requested by the business unit at the final stages of the project and that defects are detected that should have been picked up during the testing phase.

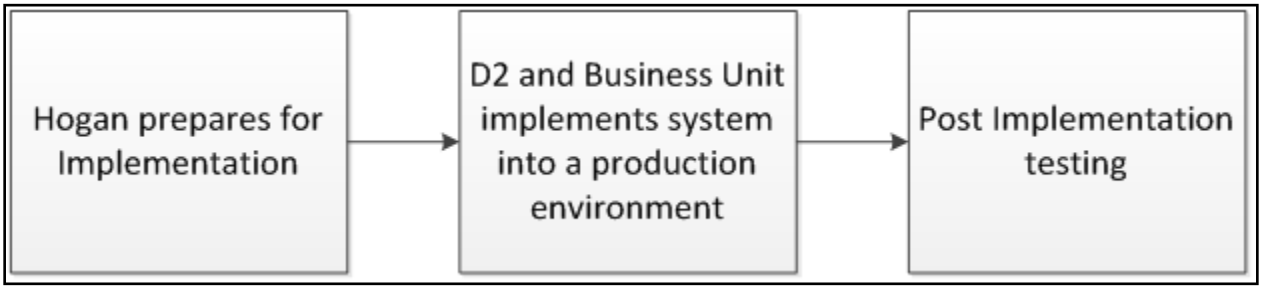


Figure 4.11 The Implementation Phase

4.4 Comparative Analysis

A comparative analysis between the current software SDLC and a SDLC that conforms to the principles of systems engineering according to the literature review was performed. Main processes and sub-processes of the current software SDLC were derived from Section 4.2 and tabulated in Table 4.2. The description of these processes between the current system and the literature review were tabulated and compared. Activities in the current system that did not align with systems engineering principles according to the literature review are indicated in bold and italic.

Table 4.1 **Comparative Analysis Results**

Systems Engineering Principle	**Recommendations to achieve principle**	**Source**
Consideration of the entire system during each activity within the SDLC	Formulating, and refining operational, functional, and performance requirements	Sweeney et al. (2011)
	Identifying and decomposing the system's functionality	Sweeney et al. (2011)
	Implementing functionality into a feasible useful product	Sweeney et al. (2011)
	Verifying the system's requirements, functionality and implementation	Sweeney et al. (2011)
	Managing inherent operational, technical and programmatic risks	Sweeney et al. (2011)
	Designing activities performed from the viewpoint of the entire life cycle	Friedman and Sage (2003)
	Using a balanced blend of methods, measurements, technologies, and processes that support the entire life cycle	Friedman and Sage (2003)
	Maintaining funding support throughout the life cycle	Friedman and Sage (2003)

	Ensuring that development activities recognise the total life cycle costs	Friedman and Sage (2003)
	Providing a thorough understanding of the system's life cycle	El-Sayar et al. (2013)
	Understanding weaknesses in the life cycle understood by modelling the life cycle according to an SE development model	El-Sayar et al. (2013)
	Monitoring project progress	College (2001)
	Evaluating and selecting alternatives	College (2001)
	Documenting data and decisions	College (2001)
	Selecting performance, functional, and design requirements, and alternative approaches to satisfy requirements based on quantitative approaches	College (2001)
Detailed definition and analysis of requirements that will achieve the users' objectives	Obtaining a clear description of the environment wherein the software will operate	Hijazi et al. (2014); Iyakutti and Alagarsamy (2011)

	Identifying all the clients of the system and collecting the raw requirements and objectives (functional and non-functional) from all points of view through observing and interviewing	Swarnalatha et al. (2014); Sommerville (2006) cited in Hijazi et al. (2014); Chomal and Saini (2014)
	Scheduling meetings with users and resources to obtain, analyse and review requirements	Chomal and Saini (2014); Iyakutti and Alagarsamy
	Developing standards and constraints in order to ensure common understanding	Swarnalatha et al. (2014); Hijazi et al. (2014)
	Defining problems clearly	Hijazi et al. (2014); Swarnalatha et al. (2014); Chomal and Saini (2014)
	Defining and analysing requirements in engineering terms and converting them into specifications for the system, its components, segments and elements	Hijazi et al. (2014); Buede (2009)
	Analysing requirements by comparing them to user or business requirements or objectives	Swarnalatha et al. (2014); Iyakutti and Alagarsamy (2011); Buede (2009)
	Quantifying performance and interface requirements	Iyakutti and Alagarsamy (2011)
	Prioritising, categorising, reviewing, and assessing	Swarnalatha et al. (2014);

	requirements according to users' objectives	Sweeney et al. (2011)
	Organising and defining requirements according to a hierarchy from high level requirements that address business requirements to low level requirements that address component requirements in a coherent and traceable manner	Swarnalatha et al. (2014); Friedman and Sage (2003)
	Performing dynamic allocation by assigning the functional and non-functional requirements to the relevant system elements	Swarnalatha et al. (2014)
	Ensuring that requirements are clear, detailed, understandable, unambiguous, comprehensive, complete and accurate	Hijazi et al. (2014); Swarnalatha et al. (2014); Sweis (2015); Khan et al. (2014); Chomal and Saini (2014); College (2001); Givens (2012)
	Ensuring that requirements specify what the system must do, how well it must do it, and how the system should be verified and validated	Iyakutti and Alagarsamy (2011); Buede (2009); Givens (2012)
	Defining and clarifying functional requirements and design constraints	College (2001); Givens (2012)
	Performing functional analysis, preliminary design and	Sweeney et al. (2011)

	planning activities by developers, documenting traceability between requirements and desired capabilities, translating development plans into requirements, and detailing how requirements should be implemented	
	Developing requirement for each function	College (2001)
	Placing equal importance on functional and non-functional aspects	Hijazi et al. (2014); El-Sayar et al. (2013)
	Resolving conflicting user requirements by choosing the most relevant requirements that will achieve the users' objectives	Hijazi et al. (2014)
	Validating and verifying requirements by reviewing the requirements with clients, prototyping according to requirements and comparing system documentation to clients' requirements and objectives	Swarnalatha et al. (2014); Sommerville (2006) cited in Hijazi et al. (2014); Li (1990)
	Negotiating the requirements with resources and end users	Chomal and Saini (2014); Iyakutti and Alagarsamy (2011)
	Performing software requirements management by keeping track of and documenting all interrelationships and dependencies of software requirements changes	Swarnalatha et al. (2014)

	Thoroughly involving developers, end users and other necessary resources during requirements development	Hijazi et al. (2014); Friedman and Sage (2003)
	Documenting requirements for future reference and ensure that it is consistent with requirements	Swarnalatha et al. (2014); Sommerville (2006) cited in Hijazi et al. (2014); Iyakutti and Alagarsamy (2011)
	Analysing cost and benefits	Iyakutti and Alagarsamy (2011)
	Reviewing preliminary project plans	Iyakutti and Alagarsamy (2011)
	Reviewing risks and contingency plans	Iyakutti and Alagarsamy (2011)
	Establishing a formal process to track and control changes to specifications	Tanrikulu and Ozcer (2011)
	Encouraging knowledge sharing between resources and users regarding technical aspects of system	Khan et al. (2014); Friedman and Sage (2003)
	Providing appropriate documentation on processes and past project successes and failures	Hijazi et al. (2014); Tanrikulu and Ozcer

		(2011); Sweeney et al. (2011)
Optimal design of the system	Ensuring extensive communication and coordination between resources and users	El-Sayar et al. (2013)
	Creating and evaluating proposed alternative designs (solutions) based on multiple criteria, including performance, schedule, cost and risk	Givens (2012)
	Modelling and evaluating preferred options and run trials	Givens (2012)
	Using preferred alternative design to manage the system life cycle	Givens (2012)
	Defining subsystems	El-Sayar et al. (2013)
	Proceeding with design once business requirement has been received and completed technology assessments have been conducted	Iyakutti and Alagarsamy (2011)
	Ensuring design is broad in perspective and every contingency is considered	Buede (2009)
	Designing the system in a logical and orderly manner	Friedman and Sage (2003)

	according to the system functional architecture	
	Choosing architectural design method and programming language early in the design phase and according to the project's need	Hijazi et al. (2014); Friedman and Sage (2003)
	Revisiting functional architecture in order to verify that the design can perform the required functions and the required level of performance	College (2001)
	Involving technical issues, customer needs, political pressures and funding in the architectural design	Friedman and Sage (2003)
	Ensuring flexible architecture	Hijazi et al. (2014)
	Ensuring design is as simple as possible and understandable	Hijazi et al. (2014); Khan et al. (2014)
	Providing complete, clear and consistent documentation of the design process free from unnecessary information	Hijazi et al. (2014); Tanrikulu and Ozcer (2011)
	Perform*ing *judgment on issues and share systems design responsibility between the development team and end user	Friedman and Sage (2003)

	Maintaining design document and specifications	Hijazi et al. (2014); Tanrikulu and Ozcer (2011)
	Providing an accurate estimation of available reusable components during requirements analysis	Hijazi et al. (2014)
	Validating and verifying design by comparing to requirements documentation, reviewing technical design; reviewing estimates on the project plan and reviewing system test plans	Iyakutti and Alagarsamy (2011)
	Testing requirements development	Tanrikulu and Ozcer (2011)
	Setting up integration plan in a manner that simplifies verification and validation	Buede (2009)
	Exploring and assessing likely alternatives to the integration order and ensure design is flexible enough to allow for changes	Buede (2009); El-Sayar et al. (2013)
Optimal integration and development of the system	Developing system once functional specification, technical design, system test plans and coding and infrastructure standards have been developed	Iyakutti and Alagarsamy (2011)

	Correcting system decomposition and well-defined components and requirements	Hijazi et al. (2014)
	Ensuring each interface and integration step throughout the life cycle supports total system functionality	Friedman and Sage (2003)
	Ensuring that the system is integrated and interfaced between different subsystems, systems, the main system and the customers	El-Sayar et al. (2013); Friedman and Sage (2003); Givens (2012)
	Ensuring all operational systems are compatible with each other	Friedman and Sage (2003)
	Excluding unnecessary specification of modules processing	Hijazi et al. (2014)
	Providing well-defined functional definitions	Hijazi et al. (2014)
	Using one developer if possible	Hijazi et al. (2014)
	Following coding standards and best practices as well as good engineering standards	Hijazi et al. (2014); Chomal and Saini (2014); Tanrikulu and Ozcer (2011)
	Avoiding repetitive code and	Hijazi et al. (2014)

	Ensuring code is understandable by reviewers	Hijazi et al. (2014)
	Reusing components where necessary	Hijazi et al. (2014)
	Using experienced programmers	Hijazi et al. (2014)
	Ensuring good quality compilers and debuggers	Hijazi et al. (2014)
	Understanding new technology before development	Hijazi et al. (2014)
	Performing integration incrementally according to the structure of the system	Hijazi et al. (2014); Li (1990)
	Ensuring correct versions and correct components are used for integration	Hijazi et al. (2014)
	Performing integration testing after each integration step	Li (1990)
	Validating and verifying the development phase by reviewing test plans and scripts, the project plan, the code and infrastructure changes, and the unit test results	Iyakutti and Alagarsamy (2011)
	Documenting software unit and system development	Hijazi et al. (2014);

		Tanrikulu and Ozcer (2011)
	Documenting roles and responsibilities	Tanrikulu and Ozcer (2011)
	Documenting unit testing results and integration test plans	Hijazi et al. (2014); Tanrikulu and Ozcer (2011)
Optimal verification and validation of the system	Proceeding with testing phase once coding scripts, test plans, and functional and technical design of the system has been developed	Iyakutti and Alagarsamy (2011)
	Ensuring that each requirement is verifiable, tested, and that the requirements documentation details the method of verification for each requirement	Hijazi et al. (2014); College (2001); Friedman and Sage (2003)
	Determining the success criteria and measures for testing early in the life cycle	Friedman and Sage (2003)
	Involving users in the verification and validation of requirements	Hijazi et al. (2014); Friedman and Sage (2003)
	Making the users the final approvers of the test outcomes	Friedman and Sage (2003)

	Setting up a test design specifications document	Tanrikulu and Ozcer (2011)
	Performing unit testing during development phase by the developer that coded the software	Li (1990); Yoon (2013)
	Using completely automated testing tools and appropriate testing techniques	Hijazi et al. (2014); Yoon (2013)
	Developing formal well-understood testing process	Hijazi et al. (2014)
	Documenting test cases for future use	Hijazi et al. (2014)
	Ensuring adequate regression testing by selecting all relevant test cases	Hijazi et al. (2014)
	Performing integration testing performed according to the structure of the system	Li (1990)
	Clearly defining test cases by describing and mapping each test case to the requirements	Li (1990); Sweeney et al. (2011)
	Reviewing, modifying, and refining test cases before test execution	Sweeney et al. (2011)

	Developing test results matrix	Sweeney et al. (2011)
	Documenting and archiving test results for future reference	Sweeney et al. (2011)
	Testing cases and data based upon system specifications	Li (1990)
	Using real-life data	Li (1990)
	Using qualified testing team	Hijazi et al. (2014); Chomal and Saini (2014)
	Ensuring sufficient number of testing resources	Hijazi et al. (2014); Chomal and Saini (2014)
	Allowing sufficient time for testers to test the entire system	Hijazi et al. (2014)
	Testing latest verified requirements	Hijazi et al. (2014)
	Conducting performance, stress and load testing	Hijazi et al. (2014)
	Verifying and validating of the testing phase performed by test results, verification of security assessments, approval for implementation and user readiness for implementation	Iyakutti and Alagarsamy (2011)
	Performing formal approval of tests	Tanrikulu and Ozcer (2011)

	Ensuring acceptance testing is performed by end users with the help of development team	Li (1990)
	Receiving formal acceptance of the system	Li (1990); Khan et al. (2014)
	Ensuring automated unit and regression testing	Parvez (2012)
	Updating regression test pack with latest test cases	Parvez (2012)
	Involving programmers with the design of unit test cases	Yoon (2013)
Successful implementation of the system	Proceeding with implementation phase once positive test results for system testing, acceptance testing, and regression testing have been obtained	Iyakutti and Alagarsamy (2011)
	Developing of a detailed implementation management plan	Tanrikulu and Ozcer (2011)
	Operating system according to standard operating procedures, formal problem management procedures and documentation procedures	Tanrikulu and Ozcer (2011)
	Ensuring that those operating the system are knowledgeable and comfortable regarding the functions	Hijazi et al. (2014); Givens (2012)

	of the system	
	Assessing performance of system using quantitative metrics	Givens (2012)
	Observing outputs of system in order to modify system where necessary	Givens (2012)
	Ensuring project team maintains the appropriate technical capabilities to gather, analyse, and recommend changes where necessary	Friedman and Sage (2003)
	Conducting implementation testing is by both the project team and end user	Friedman and Sage (2003)
	Conducting reengineering where changes in design are necessary	Friedman and Sage (2003)
	Using all data gathered during implementation testing for recommendations on future improvements	Friedman and Sage (2003)
	Documenting problems experienced during implementation	Tanrikulu and Ozcer (2011)
	Validating phase by confirming migration of the system is	Iyakutti and Alagarsamy

	according to configuration management details	(2011)
	Obtaining formal approval of project completion	Tanrikulu and Ozcer (2011)
	Establishing the post-operation review process established	Tanrikulu and Ozcer (2011)
	Designing and processing verification	Tanrikulu and Ozcer (2011)
	Inspecting documentation	Tanrikulu and Ozcer (2011)
Intensive risk management during the entire SDLC	Creating work breakdown, clearly defining and assigning duties and responsibilities to resources over time	Chomal and Saini (2014)
	Understanding project complexities and choosing resources accordingly in order to have a clear project scope	Hijazi et al. (2014); Khan et al. (2014); Patil and Yogi (2011)
	Discussing project with experts	Patil and Yogi (2011)
	Learning from past experiences	Patil and Yogi (2011)
	Conducting assessments and reviews early in the life cycle to mitigate risks	Iyakutti and Alagarsamy (2011)

	Identifying, listing, analysing, prioritising, mitigating and resolving risks and their drivers early in the project and throughout the project	Patil and Yogi (2011); Chomal and Saini (2014); Sweeney et al. (2011); Friedman and Sage (2003); Anderson and Nolte (n.d.)
	Retiring risks once successfully mitigated	Sweeney et al. (2011); Friedman and Sage (2003)
	Quantifying risks, including probability of occurrence, seriousness of impact, and assessment of impact on cost, schedule, and performance	Anderson and Nolte (n.d.)
	Determining sensitivity of risks to program assumptions and the degree of correlation among risks	Anderson and Nolte (n.d.)
	Defining and evaluating alternatives to mitigate risks	Anderson and Nolte (n.d.)
	Factoring risk into decisions on program objectives and design alternative analysis	Anderson and Nolte (n.d.)
	Tracking risks to ensure mitigation plans are effective and the potential impact on the project does not increase	Sweeney et al. (2011); Anderson and Nolte (n.d.)
	Identifying when risks become realised and become	Anderson and Nolte (n.d.)

	impediments to achievement of program goals	
	Ensuring risks are identified by any team members	Sweeney et al. (2011)
	Ensuring risks are reviewed by project manager in order to assess their relevance and probability	Sweeney et al. (2011)
	Ensuring that project performance is monitored and reported throughout life cycle	College (2001)
	Providing an adequate estimation of project time, cost, scope and resources	Hijazi et al. (2014); Khan et al. (2014); Patil and Yogi (2011); Sweis (2015); Chomal and Saini (2014)
	Assessing all resource knowledge and capabilities before scheduling	Hijazi et al. (2014); Chomal and Saini (2014)
	Estimating the schedule based on time spent on tasks and resource capabilities	Chomal and Saini (2014)
	Ensuring proper project planning and control in order to ensure realistic project schedule and budget	Hijazi et al. (2014); Khan et al. (2014); Chomal and Saini (2014)
	Ensuring good internal communication	Sweis (2015); Chomal and

		Saini (2014)
	Avoiding of assumptions	Sweis (2015)
	Verifying and validating each activity within the SDLC	Hijazi et al. (2014); Li (1990); Tuteja and Dubey (2012); Khan and Khan (2014); Tanrikulu and Ozcer (2011)
	Ensuring end users are involved throughout the entire SDLC and development activities, especially from the start of the project	Sommerville (2006) cited in Hijazi et al. (2014); Li (1990); Khan et al. (2014); Sweis (2015);Chomal and Saini (2014)
	Ensuring design changes made by end users result in timelines; revisiting requirements and project planning	College (2001)
	Monitoring project activities and milestones	Chomal and Saini (2014)
	Ensuring adequate training of resources	Khan et al. (2014); Sweis (2015); Chomal and Saini (2014); Tanrikulu and Ozcer (2011)
	Establishing regular planned meetings with all resources and end users to manage and agree on expectations, obtain	Khan et al. (2014); Chomal and Saini (2014); Iyakutti

	commitment from all involved and discuss results and status of project	and Alagarsamy (2011)
	Analysing requirements and resources to ensure realistic project expectations	Chomal and Saini (2014)
	Establishing proper processes for change management	Tanrikulu and Ozcer (2011)
	Providing a detailed implementation management plan	Tanrikulu and Ozcer (2011)
	Performing testing early in the requirements analysis phase	Tuteja and Dubey (2012); Khan and Khan (2014)
	Ensuring documentation of resource knowledge	Sweis (2015)
Involvement of resources and users from all relevant disciplines throughout the entire SDLC	Encouraging knowledge sharing among team members	Friedman and Sage (2003); Khan et al. (2014); Hijazi et al. (2014); Chomal and Saini (2014)
	Ensuring good cooperation and communication among team members	Khan et al. (2014); Hijazi et al. (2014); Sweis (2015); Chomal and Saini (2014);

		El-Sayar (2013)
	Ensuring good motivation among team members	Khan et al. (2014)
	Involving resources from the various technical and management areas as well as the end users throughout the entire life cycle, especially at the start of the project	Chomal and Saini (2014); Buede (2009)

4.5 Chapter Summary

The primary objective of this study is to identify areas within the current SDLC used by the case site that can be improved in terms of cost, schedule, and performance by performing a comparative analysis between the current software system development process and a recommended SDLC that complies with systems engineering principles. Secondly, this study aims to recommend changes to the current software system development process that will assist in the improvement in terms of cost, schedule, and performance.

This chapter presented the detailed results from the comparative analysis and identified activities in the SDLC that do not align with systems engineering principles. It is noteworthy to mention that the information provided by the high calibre experts greatly contributed to presenting the current system accurately and therefore conducting the comparative analysis effectively. The next chapter analyses and explains the findings in relation to the research objective.

CHAPTER 5 - DISCUSSION

"Discussion is an exchange of knowledge; an argument an exchange of ignorance."

~ Robert Quillen

5.1 Introduction

This chapter analyses and discusses the findings presented in Chapter 4 in relation to the research objectives and in conjunction with the literature reviewed in Chapter 2. The aim of this chapter is to find answers to the research question:

> ***How can the software SDLC be improved in terms of cost, schedule, and performance for a credit scoring system using systems engineering principles?***

It was found that the current software SDLC could be improved in terms of cost, schedule and performance. This is outlined in Section 5.3 based on the recommendations detailed in Section 5.2.

5.2 Comparative Analysis Discussion

This section discusses the results from the comparative analysis. The aspects of the current system that were found to be inefficient according to systems engineering principles and the recommendations to align these aspects with systems engineering principles are detailed.

5.2.1 Definition and analysis of requirements

Appropriate documentation on processes and past project successes and failures: Past project successes and failures are discussed among resources during the requirements

analysis phase. However, knowledge on processes and past project details is not gathered from documentation but rather from the discussion regarding the resources' past experiences. Lessons learnt from past projects are not documented. It is thus recommended that each resource document the processes as well as the successes and failures they experience during the project (Hijazi et al., 2014; Tanrikulu & Ozcer, 2011; Sweeney et al., 2011). These documents can be used for future reference to improve future projects and avoid similar pitfalls. Knowledge would therefore not be lost in cases of staff turnover.

Estimation of schedule based on time spent on tasks and resource capabilities: It is recommended that the project manager consider the current resource knowledge and capabilities in estimating the project schedule.

5.2.2 Optimal design of the system

Flexible architecture: Currently, the system is designed according to the chosen solution. However, flexibility is not built into the design to allow for alternative design solutions should an one prove to be best later in the life cycle. It is recommended that flexibility be built into the system that would allow for the most likely alternative design solutions (Hijazi et al., 2014).

Perform judgment on issues and share systems design responsibility between the development team and end user: Although the design is reviewed by the end user, responsibility is not shared among the systems analyst, developer and end user. It is recommended that responsibility be equally shared thus allowing the end user to obtain a detailed understanding of the system from the start of the project, which will in turn increase the probability of user acceptance later in the life cycle (Friedman & Sage, 2003).

Accurate estimation of available reusable components during requirements analysis: It is recommended that the developer estimate the available reusable components during this phase as these could have a significant impact on the project timelines and cost (Hijazi et al., 2014). The use of reusable components will shorten development timelines. The project manager would then be able to more accurately estimate project timelines.

Set up integration plan in a manner that simplifies verification and validation: The developer currently integrates the system according to the structural design of the system as

specified in the requirements documents. However, an official integration plan is not set up. It is recommended that the developer set up an integration plan according to the functional architecture while at the same time considering the validation and verification of the system (Buede, 2009). This will result in simplified integration testing and thus reduced development timelines.

Explore and assess likely alternatives to the integration order and ensure design is flexible enough to allow for changes: It is recommended that the developer explore and assess likely alternatives to the integration order and ensure that the design is flexible enough to allow for an alternative integration order during development (Buede, 2009; El-Sayar et al., 2013).

5.2.3 Optimal integration and development of the system

Use of experienced programmers: Not all projects currently use experienced programmers (developers). Programmers are selected based on availability. As lack of documentation and staff turnover results in lost knowledge and new programmers often lack the necessary knowledge, Hijazi et al. (2014) suggest that programmers should be experienced.

It is therefore recommended that development processes and knowledge be adequately documented by developers for each project. It is also recommended that new programmers shadow experienced programmers during the first few projects until the new programmers gain confidence in the development process.

Understanding of new technology before development: It is currently not ensured that new technology is well understood before development takes place. Hijazi et al. (2014) suggest that new technology should be thoroughly understood before development takes place. It is therefore recommended that adequate training of new technology takes place before project schedule and development.

Documentation of software unit and system development: It is recommended that unit development and system development be documented in detail for future reference (Hijazi et al., 2014; Tanrikulu & Ozcer, 2011). This will assist in verification, validation and knowledge gathering regarding lessons learnt.

Documentation of unit testing results and integration test plans: It is recommended that test plans and test results be documented in detail to simplify verification, validation, and knowledge accumulation (Hijazi et al., 2014; Tanrikulu & Ozcer, 2011).

5.2.4 Optimal verification and validation of the system

Involve users in the verification and validation of requirements: Hijazi et al. (2014) and Friedman and Sage (2003) recommend that the end users should be involved in the verification and validation of requirements. Currently, the users are solely involved during the requirements analysis and user acceptance testing phases. It is recommended that the users be involved during unit, functional, and complete system testing. This is expected to simplify the user acceptance testing as the users will already be familiar with the testing details.

Make users the final approvers of the test outcomes: It is recommended that the users be assigned as the final approvers of the test outcomes of unit, functional, and complete system testing as this will increase the probability of user acceptance testing (Friedman & Sage, 2003).

Use of complete automated testing tools and appropriate testing techniques: Hijazi et al. (2014) and Yoon (2013) recommend that complete automated testing tools and appropriate testing techniques should be used during testing. Functional testing is currently a manual process in which 20 to 30 customer profiles are manually created, manipulated and tested according to test case scenarios. This process is time consuming and costly as it requires on average six test analysts to perform functional testing. Verification is additionally compromised as the entire customer data base is not tested.

It is recommended that functional testing be improved by executing test cases on a large set of customer data in order to adequately cover the software change that was made. It is recommended that the following procedure be followed to achieve this:

- Extract a sample of approximately 10 to 15 per cent of customer account data from the production environment for test cases.
- Extract data from the different data warehouses in order to obtain the necessary field information.

- Set up filtering requirements in order to have a variety in spread and selection of data. This will result in an increase in test coverage as well as testing a wider variety of scenarios.
- As testing is performed in an uncontrolled environment (which means that the testers will be able to view actual customer data), desensitise and depersonalise the data by breaking the link from the production profile to the test data. Re-match the customer account data by means of algorithms when it needs to be sent to the credit bureau to obtain the customers' credit information. Use the same logic to depersonalise the records once it is receive back from the credit bureau.
- Automate functional testing by running mathematical algorithms through the entire sample.

Adequate regression testing by selecting all relevant test cases: It is recommended that a test pack be created that adequately covers the software changes that were made (Hijazi et al., 2014). The regression test pack should be continuously updated by adding the same test data that is recommended for functional testing to the test pack (Parvez, 2012). This test pack will thus contain a sample customer data base that is representative of the customer data base in the production environment. The tester is therefore able to perform regression testing on this entire sample instead of only one customer. The tester will also be able to perform negative testing on this sample in order to see whether the system is producing the expected results.

Use of real life data: It is recommended that functional and regression testing be performed on a representative sample of the production customer data base (Li, 1990).

5.2.5 Successful implementation of the system

Ensure that those operating the system are knowledgeable and comfortable regarding the functions of the system: Hijazi et al. (2014) and Givens (2012) recommend that those operating the system should be knowledgeable and comfortable regarding the functions of the system. It is therefore recommended that users are more involved in design, development, and testing in order to ensure that they are comfortable and knowledgeable regarding the functions of the system.

Implementation testing is conducted by both the project team and end user: Currently, implementation testing is conducted by the project team. It is recommended that the users be involved to reduce the probability of defects and rework (Friedman & Sage, 2003).

Use all data gathered during implementation testing for recommendations on future improvements: It is recommended that data gathered be analysed and documented (Friedman & Sage, 2003). It is also suggested that recommendations for future projects be documented during the post-implementation review meeting.

Documentation of problems experienced during implementation: It is recommended that problems and successes experienced during implementation be documented in order to avoid similar pitfalls and make improvements during future projects (Tanrikulu & Ozcer, 2011).

Inspection of documentation: It is recommended that all documentation be reviewed by the project team, project manager and end users to avoid defects (Tanrikulu & Ozcer, 2011).

5.2.6 Intensive risk management during the entire SDLC

End users involved throughout the entire SDLC and development activities especially from the start of the project: It is recommended that the end users should be thoroughly involved in all development activities within the life cycle (Sommerville, 2006 cited in Hijazi et al., 2014; Li, 1990; Khan et al., 2014; Sweis, 2015; Chomal & Saini, 2014). It is therefore recommended that users play a more significant role during the design, development, and testing phases in order to avoid defects and rework and improve the probability of user acceptance.

Adequate training of resources: It is recommended that resources should be adequately trained in order to perform their work as efficiently and effectively as possible (Khan et al., 2014; Sweis, 2015; Chomal & Saini, 2014; Tanrikulu & Ozcer, 2011). It is therefore recommended that resource training be supplemented by documentation on system processes, successes and failures.

5.3 Comparative Analysis Implications

The current SDLC has a 90 day schedule and requires on average 20 resources per project of which six are test analysts. Forty per cent of projects at the case site were not successfully completed during 2015 according to the end users' needs within the required timelines. These projects were therefore rescheduled for the following annual release dates. It is suggested that once the recommendations are implemented, the project manager should monitor the projects in the subsequent six months in terms of project schedule, resources required per system area, and percentage of projects successfully completed.

Table 5.1 details how the recommendations are expected to improve cost, quality and schedule:

Table 5.1 **Recommendations**

Recommendation	**How recommendation is expected to improve cost, schedule and quality**
Appropriate documentation on processes and past project successes and failures	Reduces the probability of defects and rework as pitfalls are avoided. Past successes on similar projects are used to develop the system efficiently.
Estimation of schedule based on time spent on tasks and resource capabilities	Project manager is able to more accurately schedule the project and assign resources therefore increasing the probability of delivering the project on time with the desired capabilities.
Flexible architecture	Reduces rework required in the event that an alternative design is required at a later stage of the life cycle.
Perform judgment on issues and share systems design responsibility between the development team and end user	Results in users being involved throughout the SDLC thus improving the likelihood of user acceptance at the end stage of the SDLC and reducing the likelihood of rework in the event that the system does not completely meet the users' requirements.
Accurate estimation of available reusable components during requirements analysis	Project manager is able to schedule the project more accurately, which increases the likelihood of delivering the project within the required timeline with the desired capabilities.
Set up integration plan in a manner that simplifies verification and validation	Reduces schedule thus improving cost and quality by simplifying the verification and validation process.
Explore and assess likely alternatives to the	Reduces rework required in the event that an alternative integration order and

integration order and ensure design is flexible enough to allow for changes	design is required at a later stage of the life cycle.
Use of experienced programmers	Reduces the likelihood of defaults and rework.
Understanding of new technology before development	Reduces schedule and the likelihood of defects as developers do not spend time during the project life cycle trying to understand the new technology and develop the system accordingly.
Documentation of software unit and system development	Improves quality of verification and validation. Schedule and cost is reduced as time is not wasted trying to understand system activities when revisiting previous phases or projects.
Documentation of unit testing results and integration test plans	Improves quality of verification and validation. Schedule and cost is reduced as time is not wasted trying to understand system activities and results when revisiting previous phases or projects.
Involve users in the verification and validation of requirements	Reduces defaults and rework as the likelihood of user acceptance is increased.
Make users the final approvers of the test outcomes	Reduces defaults and rework as the likelihood of user acceptance is increased.
Use of complete automated testing tools and appropriate testing techniques	Reduces the number of testing analysts required. Improves quality by testing a larger customer data base.
Adequate regression testing by selecting all relevant test cases	Improves verification quality by testing a larger number of customer profiles.

Use of real life data	Improves verification quality by testing a larger number of customer profiles. Thus reducing probability of defects and rework.
Ensure that those operating the system are knowledgeable and comfortable regarding the functions of the system	Keeps users involved throughout the SDLC thereby increasing the likelihood of user acceptance at the end of the life cycle.
Implementation testing conducted by both the project team and end user	Increases the likelihood of user acceptance thereby reducing the likelihood of defects and rework.
Use all data gathered during implementation testing for recommendations on future improvements	Avoids similar pitfalls and uses lessons learn to improve schedule and quality of future projects thereby reducing project costs.
Documentation of problems experienced during implementation	Avoids similar pitfalls and uses lessons learn to improve schedule and quality of future projects thereby reducing project costs.
Inspection of documentation	Ensures accuracy of documentation on the system to improve verification and validation quality and schedule.
End users involved throughout the entire SDLC and development activities especially from the start of the project	Increases the likelihood of user acceptance at the end of the life cycle as end users are involved in the verification and validation at each phase of the life cycle.
Adequate training of resources	Adequate training of resources before the project commences reduces the likelihood that defects are built into the system thus reducing the likelihood of rework. It also improves the quality and schedule of the project as resources do not spend time during the life cycle trying to understand aspects of the system.

5.4 Chapter Summary

The intention of the comparative analysis was to seek ways to improve the current software SDLC in terms of cost, schedule and performance. This chapter discussed the results obtained from the comparative analysis. Inefficiencies in the current SDLC were identified that did not align with SE principles and recommendations were made to improve the current SDLC by aligning it with these principles.

CHAPTER 6 – CONCLUSION AND RECOMMENDATIONS

“I think and think for months and years. Ninety-nine times the conclusion is false. The hundredth time I am right.”

~ Albert Einstein

6.1 Conclusions

The objective of this research was to determine whether the current software SDLC can be improved in terms of cost, schedule, and performance. The current system was analysed by conducting face-to-face semi-structured interviews and using visual sense making techniques.

An in-depth literature review was conducted in order to investigate SE principles and to find recommendations on how to align projects with SE principles. A comparative analysis was subsequently conducted between the current software SDLC and the recommendations obtained through the literature review.

Areas of inefficiencies in the current software SDLC were found that did not fully align with systems engineering principles or were partly aligned with systems engineering principles but could be improved. Insight obtained from the literature review was used to make recommendations to improve these areas of inefficiencies in terms of cost, schedule and performance.

As stated in the literature review, project success is generally determined by meeting the end users’ objectives in terms of time, cost and quality. Additionally, these three aspects are stated in the literature review to be the main causes of project success or failure. It is therefore necessary to improve the current SDLC in terms of these aspects.

6.2 Implications of the Research

The study can be applied to various systems development life cycles in a variety of industries. It contributes to a good starting point for developing a SE framework that can be applied to software SDLCs within multiple industries.

6.3 Limitations

Limited documentation on the current software SDLC was available. The research was therefore limited to the opinions of the current resources and experts from the various disciplines. The research was additionally limited to a single case site at a banking institution and did not incorporate SDLC aspects of other financial institutions.

6.4 Recommendations for Future Work

The research concentrated on improving the software SDLC within the case site. There are a few areas that require further study:

1. Examining the impact and true value of the recommendations made to the case site.

2. Conducting a replication study at other financial institutions in order to establish a general framework for applying SE principles to software development processes.

BIBLIOGRAPHY

Anderson, N. and Nolte, W. (n.d.) *Systems engineering principles applied to basic research and development.* Albuquerque: Air Force Research Laboratory.

Boehm, B. (2000) *Spiral development: experience, principles and refinement.* Pittsburg: Carnegie Mellon University.

Buede, D. M. (2009) *The engineering design of systems – Models and methods.* New Jersey: John Wiley & Sons, Inc.

Chomal, V. S. and Saini, D. J. R. (2014) Cataloguing most severe causes that lead software projects to fail. *International Journal on Recent and Innovation Trends in Computing and Communication,* vol.2, no.5, pp. 1143 – 1147.

Cohen, L., Manion, L. and Morrison, K. (2007) *Research methods in education.* 6th ed. New York: Routhledge.

Department of Defense Systems Management College (2001) Systems engineering fundamentals. Belvoir: Defense Acquisition University Press.

Coughlan, M., Cronin, P. and Ryan, F. (2007) Step-by-step guide to critiquing research. Part 2: qualitative research. *British Journal of Nursing,* vol.16, no.11, pp. 738 – 744.

Coyne, I. T. (1997) Sampling in qualitative research. Purposeful and theoretical sampling: Merging or clear boundaries? *Journal of Advanced Nursing,* vol.26, no.3, pp. 624.

Creswell, J. W. (2007) *Qualitative inquiry and research design. Choosing among 5 approaches.* 2nd ed. Thousand Oaks: Sage Publications.

Deming, W. (1966) *Some theory of sampling.* New York: Dover Publications.

Denzin, N. K. and Lincoln, Y. S. (1994) *Handbook of qualitative research.* Thousand Oaks: Sage Publications.

El-Sayar, N. M., Afefy, I. H. A. and El-kamash, A. (2013) Addressing problems by systems engineering methods, techniques, and tools-model framework. *International Journal of Innovative Research in Science, Engineering and Technology,* vol.2, no.12, pp. 7373 – 7376.

Flick, U. (2009) *An introduction to qualitative research.* 4th ed. Thousand Oaks: Sage Publications.

Friedman, G. and Sage, A. P. (2003) Case studies of systems engineering and management in systems acquisition. *Systems Engineering*, 22 September, pp. 90 – 96.

Givens, A. (2012) A systems-based approach to intelligence reform. *Journal of Strategic Security,* vol.5,no.1, pp. 68 – 69.

Golafshani, N. (2003) Understanding reliability and validity in qualitative research. *The Qualitative Report,* Vol.8, pp. 597 – 607.

Hari, A., Shoval, S. and Kasser, J. (2008) *Conceptual design to cost: A new systems engineering tool.* INCOSE.

Hijazi, H., Alqrainy, S., Muaidi, H. and Khdour, T. (2014) Risk factors in software development phases. *European Scientific Journal,* vol.10, no.3, pp. 213 – 232.

Honour, E. C. (2004) *Understanding the value of systems engineering.* Penascola: INCOSE.

Howard, K. (2011) *Early education from a parental perspective : A qualitative study.* Michigan: University of Michigan.

Ismail, M. S. (2012) *Analysis of project management techniques within software engineering in the financial industry.* Johannesburg: University of Johannesburg.

Iyakutti, D. K. and Alagarsamy, D. K. (2011) Software development life cycle standards for banking and financial services IT industry. *International Journal of Wisdom Based Computing,* Vol.1,no.3, pp. 146 – 167.

Johnson, G. B. R. (1977) *Statistical concepts and methods.* New York: John Wiley & Sons.

Kasser, J. (2010) *Holistic thinking and how it can produce innovative solutions to difficult problems.* Stockholm: INCOSE.

Kasser, J. E. (2007) *Eight deadly defects in systems engineering and how to fix them.* Mawson Lakes: INCOSE.

Kaur, R. and Sengupta, J. (2011), Software process models and analysis on failure of software development projects. *International Journal of Scientific & Engineering Research ,* vol.2,no.2, pp. 1.

Khan, K., Qadri, S., Ahmad, S., Siddique, A.B., Ayoub, A. and Saeed, S. (2014) Evaluation of PMI's risk management framework and major causes of software development failure in software industry. *International Journal of Scientific & Technology Research,* vol.3,no.11, pp. 120 – 123.

Khan, M. E. and Khan, F. (2014) Importance of Software Testing in Software Development Life Cycle, *International Journal of Computer Science,* vol.11, no.2, pp. 120 – 123.

Koerber, A. and McMichael, L. (2008) Qualitative sampling methods: A primer for technical communicators. *Journal of Business and Technical Communication,* vol.22, no.4, pp. 463 – 467.

Leedy, P. D. and Ormrod, J. E. (2005) *Practical research: Planning and design.* 8th ed. New Jersey: Prentice Hall.

Li, E. Y. (1990) Software testing in a system development process: A life cycle perspective. *Journal of Systems Management,* vol.41,no.8, pp. 23 – 31.

Lincoln, Y. and Guba, E. (1985) *Naturalistic inquiry.* Beverley Hills: Sage Publications Inc.

London, B. N. (2012) *A model-based systems engineering framework for concept development.* Boston: Massachusetts Institute of Technology.

Luo, L. (n.d.) *Software testing techniques – Technology maturation and research strategy,* Pittsburgh: Carnegie Mellon University.

Maier, M. W. (1999) *Architecting principles of systems-of-systems.* Chantilly: John Wiley & Sons Inc.

Martin, J. N. (2004) *The seven samurai of systems engineering: Dealing with the complexity of 7 interrelated systems.* Chantilly: INCOSE.

Mathur, S. and Malik, S. (2010) Advancements in the V-model. *International Journal of Computer Applications,* 1(12), pp. 29 – 33.

McMurtrey, M. (2013) A case study of the application of the systems development life cycle (SDLC) in the 21st century health care: Something old, something new? *Journal of the Southern Association for Information Systems,* vol.1, no.1, pp. 1.

Miles, M. B. and Huberman, A. M. (1994) *An expanded sourcebook: Qualitative data analysis.* 2nd ed. Thousand Oaks: Sage Publications.

Mooz, H. and Forsberg, D. K. (2004) *Clearing the confusion about spiral/evolutionary development,* INCOSE, pp. 4.

Mudavanhu, T. B. (2013) *Towards a sustainable framework for application of the systems engineering approach (SEA) in non-traditional implementation areas.* Johannesburg: University of Witwatersrand.

Munassar, N. M. A. and Govardhan, A. (2010) A comparison between five models of software engineering. *International Journal of Computer Science,* vol.7, no.5, pp. 94 – 100.

Myers, G. J. (2004) *The art of software testing.* New Jersey: John Wiley & Sons, Inc.

Okafor, E. F. O. (2011) Challenges and solution of software engineering and development: A review. *International Journal of Current Research,* vol.3, no.5, pp. 65 – 66.

Parvez, A. W. M. M. (2012) An efficient model for mobile application regression test for agile scrum software development. *Advances in Computer Science and its application,* vol.2, no.2, pp. 339 – 343.

Patil, M. V. and Yogi, A. N. (2011) Importance of data collection and validation for systematic software development process. *International Journal of Computer Science & Information Technology,* vol.3, no.2, pp. 260 – 262.

Patton, M. (1987) *How to Use Qualitative Methods in Evaluation.* 2nd ed. Newbury Park: Sage Publications.

Patton, M. (1990) *Qualitative evaluation and research method.* 1st ed. Beverley Hills: Sage Publications.

Pickvance, C. (2005) *The four varieties of comparative analysis: the case of environmental regulation.* Kent: University of Sussex.

Ragunath, P. K., Velmourougan, S., Davachelvan, P., Kayalvizhi, S. and Ravimohan, R. (2010) Evolving a new model (SDLC Model-2010) for software development life cycle (SDLC). *International Journal of Computer Science and Network Security,* vol.10, no.1, pp. 112 – 118.

Rajkumar, G. and Alagarsamy, K. (2013) The most common factors for the failure of software development project. *The International Journal of Computer Science and Applications,* vol.1, no.11, pp. 1.

Rather, M. A. and Bhatnagar, V. (2015) A comparative study of sftware development life cycle models. *International Journal of Application or Innovation in Engineering and Management,* vol.4, no.10, pp. 23 – 28.

Royce, W. W. (1988) *Managing the development of large software systems.* Washington: IEEE Computer Society Press.

Shrinivasan, Y. B. (2010) *Supporting the sensemaking process in visual analytics.* Eindhoven: University of Technology.

Sommerville, I. (2006) *Software engineering.* 8th ed. Addison Wesley.

Strauss, A. and Corbin, J. (1990) *Basics of qualitative research: Grounded theory procedures and techniques.* Newbury Park: Sage Publications.

Swarnalatha, K. S., Shrinivasan, G.N., Dravid, M., Kasera, R. and Sharma, K. (2014) A survey of software requirement engineering for real time projects based on customer requirement. *International Journal of Advanced Research in Computer and Communication Engineering,* vol.3, no.1, pp. 5045 – 5050.

Sweeney, R. L., Hamman, J. P. and Biemer, S. M. (2011) The application of systems engineering to software development: A case study. *Johns Hopkins APL Technical Digest,* vol.29, no.4, pp. 329 – 336.

Sweis, R. J. (2015) An investigation of failure in information systems projects: The case of Jordan. *Journal of Management Research,* vol.7, no.1, pp. 173 – 181.

Tanrikulu, Z. and Ozcer, T. (2011) Standardization of information systems development processes and banking industry adaptations. *International Journal of Software Engineering & Applications,* vol.2, no.2, pp. 1 – 3.

Tuteja, M. and Dubey, G. (2012) A research study on the importance of testing and quality assurance in software development life cycle (SDLC) models. *International Journal of Soft Computing and Engineering,* vol.2, no.3, pp. 251 – 156.

Walk, K. (1998) *How to write a comparative analysis,* http://writingcenter.fas.harvard.edu/pages/how-write-comparative-analysis, Accessed 5 Dec 2015.

Wengraf, T. (2001) *Qualitative research interviewing.* London: Sage Publications.

Wilkinson, D. and Birmingham, P. (2003) *Using research instruments, a guide for reserachers.* 1st ed, London: RoutledgeFalmer.

Yoon, H. (2013) Effort reduction of unit testing by supporting CFG generation and its test design, *International Journal of Smart Home,* vol.7, no.5, p. 127.

APPENDICES

APPENDIX A – *Interview Introductory letter*

“Improvement of the Software Systems Development Life Cycle of the Credit Scoring Process at a financial institution through the application of Systems Engineering”

Feb 4, 2015

Dear Sir/Madam

We are writing to ask for your assistance with a study we are conducting focusing on the improvement of the current software systems development life cycle of the credit scoring process. More specifically the purpose of this research is to seek ways to improve the current SDLC in terms of cost, schedule and performance.

A comparative analysis between the current system and the ideal system according to systems engineering principles will be conducted. In order to perform the comparative analysis, a detailed understanding of the current system will be required. Please regard this letter as an invitation to participate in this research in your capacity as a subject area expert. The study will be conducted using face-to-face in depth semi-structured interviews.

Thank you for taking the time to read this letter. Please provide feedback by 11 February 2015. We look forward to working with you. For further information please feel free to contact the undersigned researcher directly.

Sincerely,

Ms. Nadia Meyer	**Ms. Bernadette Sunjka**
Researcher: *Msc. Candidate*	Advisor (Post grad Coordinator)
Student # 870445	Senior Lecturer
Cell: +27 82 555 1585	Cell: +27 11 717 7367
Email: nadiameyer50@gmail.com	Email: Bernadette.sunjka@wits.ac.za

Faculty of Engineering and Built Environment

School of Mechanical, Industrial and Aeronautical Engineering

Private Bag 3, University of Witwatersrand, Johannesburg, 2050, South Africa

APPENDIX B – *Interview Questions*

- Describe the area you are involved in
- Describe the processes involved this area and their order
- Describe the inputs and outputs of the different processes
- Explain how each process works
- Provide detail on resources, departments and timelines involved with each process
- What do you perceive to be the problems in the current SDLC?
- What do you perceive are the strengths within the current SDLC?
- How could the problems be addressed?

APPENDIX C – *Interview Transcripts*

Interview transcript - Participant 1 (Project Manager)

Researcher: Describe the area you are involved in

Participant 1: I am the project manager responsible for the entire project development that occurs when a change to the current credit scoring logic is requested. Credit scoring logic usually includes the credit origination scorecard, limit and affordability calculations, pre-bureau rules and post-bureau rules. When these changes are requested, a project is logged and initiated in order to develop a new software system that contains the software changes.

Researcher: Describe the processes involved this area and their order

Participant 1: The current software development process that is involved in making a software change is developed according to a traditional V-model design. The V-model has a corresponding test activity for every development activity. Analysis and design of tests begin during the corresponding development activity for each level. The design of the V-model allows for risk management.

Researcher: What type of risks do you address?

Participant 1: There are various risk factors involved in testing such as risks relating to schedule, requirements, human resources, and quality. Risks in schedule are related to unrealistic project schedules requested by the relevant business unit. Therefore expectations are managed during the requirements analysis phase in order to avoid this as far as possible. The project could face several risks due to user requirements. This could be due to lack of clarity in user requirements, ambiguous requirement definitions, changes in the requirements or unrealistic requirements. Testers are involved in reviewing the specifications as soon as possible in the lifecycle, in order to pick up defects in the specifications early on. Factors such as unrealistic schedules, lack of resources, and frequent requirement changes could compound the risk of poor quality of the system. Therefore not only are business expectations managed as far as possible, but also requirements analysis is performed in detail in order to develop an optimal and realistic requirements definition. The project could face risks if there is a lack of human resources available with the necessary skills required in the project. These issues are therefore addressed during impact analysis, project sizing and scheduling meetings. Quality is

another risk factor of a project. Therefore testing is conducted at all levels of software development in order to ensure that defects are not present at a particular level and then later built into larger parts of the software system. Testing is required to test the developed software in order to ensure that the level of quality is on par with the specifications, to provide information for decision making, to prevent defects, and to mitigate risk. Testing involves analyzing the tests, designing test cases, implementing the tests, executing the tests and finally, comparing the test results. Various types of testing are performed at all stages of the V-model to test for changes that were made to the software. The test analysis and design begins early in the development process followed by testers reviewing the development documentation. Testing is already performed early in the lifecycle in order to ensure optimal quality of the process. This approach to testing is costly and time-consuming but ensures that the correct results are achieved by the end of the project.

Researcher: What is the V-model?

Participant1: The V-model is a software development model which describes and specifies how software development should take place. The activities in the SDLC are carried out according to the V-model design. The V-model design ensure that testing is performed for every step. Thus testing is performed early.

Researcher: Why is testing early in the lifecycle so important?

Participant1: If tests are designed as early as possible, defects in the specifications will be found early in the process when they are still inexpensive to fix. If the testing is done too late in the lifecycle, time and effort may well be saved but testing quality could be compromised as defects could be found much later when they are more expensive to fix. In addition, the defects in the higher levels, such as the requirements specification, will be found late. These defects are the most critical and important as defects in the higher levels will be built into the lower levels of the V-model. Another reason why the tests are designed early according to the V-model is to ensure that the order in which the system should be put together for testing is defined during the design phase. The order of software development is specified before it is built. Testers are thus not constrained by the order in which software is built as tests are designed early in the lifecycle according to the specifications and do not rely on the software to be built first in order to initiate testing. This allows the integration of the system to be known early therefore the system can be built based on the known integration. This can greatly reduce development time later on as development activities and verification of these activities are performed early. Testing time is

reduced as testing activities are broken down to be performed at different stages of the life cycle. For example, the first integration can be built and tested first before continuing to integrate more parts. This also allows integration testing to be performed simultaneously to development of the system, thus saving time but not necessarily effort. When problems are picked up, they are immediately fixed before proceeding. If testing is only done once at the end of the process when the entire system has been put together, it might be difficult to find the cause of the defects as the cause could be at any of the levels in the V-model. In summary, the benefit of designing tests as early as possible is that quality is built in, costs are reduced, and time is saved as fewer defects are found. Therefore quality is built into the software development process.

Researcher: How is testing performed?

Participant1: Testing is performed according to specifications. The specifications specify what the correct results of the testing should be. Specifications are documented in order to ensure that there is no misunderstanding or lack of clarity. Specifications are designed by taking testing into consideration. Knowing how the tests will be performed assists in developing the specifications for development. The specifications therefore simplify the process of setting up test cases. Testing reveals defects in code, parts of the system, the system as a whole, or the user's view of the system. Specifications are compiled in all levels of the V-model, from a business requirement specification to a specification for the code. The tester, quantitative analyst or developer performs a test comparison that detects the differences between the actual test results and the expected test results by using the business requirement specifications. The business requirements are validated by discussing them with users in the business unit and comparing them against the knowledge of the business unit's working practices. If a defect is detected at any stage of testing, the defect is reported and subsequently resolved by recoding the software. A new version of the software is released containing the fixed defect. Before being able to continue to the next phase of testing, re-testing is performed on the latest software version in order to ensure that the defect has been fixed correctly and the rest of the system is still working as per the requirements. Re-testing is executed in the same environment and using the same test cases.

Researcher: Why are the same test cases used?

Participant1: The reason that all of the test cases are executed again and not just the test case pertaining to the defect is that the fixed defect could possibly introduce new unexpected defects

elsewhere in the system. This might however still not be picked up as only one part of the system has been tested. Regression testing is finally performed in order to address this issue.

Researcher: What happens after re-testing has been performed?

Participant1: Regression testing is performed. Regression testing tests whether the changes made to a particular part of the software has not caused secondary problems elsewhere in the software system. This ensures that modifications in the software or environment still meet the original requirements without causing unintended side effects in the system. It is performed every time changes to the software or environment are made. Regression testing is performed at all levels in the V-model in order to ensure that the requirements are met at all levels.

Researcher: How is regression testing performed?

Participant1: Regression testing is automated through the use of mathematical algorithms that are applied to a set of test cases. The test cases currently consist of only one customer profile with account data.

Researcher: Describe the inputs and outputs of the different processes

Participant1: There are main phases involved in the development process: Requirements analysis phase, design phase, development phase, testing phase and implementation and maintenance phase. These phases occur sequentially.

Researcher: Explain how each process or phase works and provide detail on resources, departments and timelines involved with each process

Participant 1: The requirements analysis phase as represented is where the requirements of the business are determined and analysed. The business unit defines and analyses their own needs and transforms it into a requirements specification. Once the business requirements are developed by the relevant business unit, it is submitted to analytics team in the form of a document called a Business Requirement Specification (BRS). This allows the business unit to formally request the changes. The BRS contains specifications of the development required. The requirements are logged as a project and a project manager and project team is assigned.

Researcher: Who does the project team consist of?

Participant 1: The project team consists of the resources from the various disciplines involved in the SDLC such as the analysts, developers, project manager, divisional managers, and

testers. The BRS is reviewed by the project team in order to determine if the information provided is sufficient for development. The project team also determines whether the requirements (functional and non-functional) are realistic and whether it will achieve the desired end objective of the business unit (stakeholders). Communication with the business unit requesting the change is initiated in order to confirm the details of the requirements and understand and define the problems and objectives. Should the information provided be insufficient, the BRS is sent back to the business unit for amendments to be made. Afterwards the project team will once again review the specification. Once the project team finds the information provided to be sufficient, it is submitted for a high level impact assessment.

Researcher: Who is involved in the high level impact assessment?

Participant 1: The project team, project manager and the business unit (end users) is involved in the high level impact assessment. During the high level impact assessment the project is sized, dependencies and impact on other areas are determined, and additional information is requested if required. Standard language is defined in order to ensure that all the resources and stakeholders have a mutual understanding. The implications of changes made, the external and internal influences on the system, and the probability of successful completion of the project within budget and schedule are determined. Risks are identified, listed, quantified, analysed and prioritised. The probability of risk occurrence, seriousness of impact, assessment of impact on cost, schedule, and performance, sensitivity to assumptions, and degree of correlation amongst risks are determined. Alternatives to mitigate risks are defined and evaluated. Preventative and contingency measures are set in place. Risks are tracked throughout the SDLC to ensure that mitigation plans are effective and the potential impact on the project does not increase. Agreement is obtained on timelines and funding required. Commitment by all involved is obtained. After the impact assessment, the project is scheduled. During scheduling aspects such as the business priority, areas involved, dependencies on other projects, and whether a Functional Requirement Specification (FRS) is required are determined by the project manager and project team members. The Functional Requirement Specification specifies the changes required on a functional level. After the project has been scheduled, the project scope is confirmed. The project manager, project team members and business unit is involved in the scoping session. The business unit presents what the project is about and gives an overview of the change and requirements involved. The project is once again examined in order to determine if anything has changed. If changes have been made, the process is repeated from the impact assessment phase. Otherwise, the requirements are finalised and whether a FRS is

required is finally determined. Should a FRS be required, the business analyst develops the FRS.

Researcher: How is the phase verified and validated?

Participant 1: This phase is validated and verified by analysing and reviewing the requirements by comparing it to the business unit objectives, documenting the requirements definition, analyzing the cost and benefits, reviewing the preliminary project plan, and reviewing risks and contingency plans.

Researcher: What are the outputs of the phase?

Participant 1: The outputs of this phase are the approved project scope and objectives, and initial estimation of cost and benefit, project priority, a preliminary project plan, a tentative release date, approved business requirements, assessment of all technology involved, and a project life cycle assignment. The deliverables are the BRS and FRS.

Researcher: How long does this phase take to complete?

Participant 1: The requirements phase takes approximately 2-4 weeks to complete.

Researcher: Proceed..

Participant 1: The analysis and design phase is initiated once the outputs of the requirements phase have been concluded. The analysis and design phase as represented in is where detailed requirements are transformed into a complete detailed design of the system by focusing on how to deliver the required functionality. The systems analyst produces a Systems Requirements Specification (SRS) that specifies the changes required to the current system. It is ensured that the SRS is detailed, clear and accurate. The technical design is verified and validated by ensuring that the functional architecture can perform the required functions and required level of performance according to the BRS and FRS. The resultant designed architecture is approved by the Hogan Technical Board (HTB). The Hogan Channel Design Forum (CDF) then approves the required changes to the business functions. Parallel to this, the test analyst proceeds to develop the test requirements and design the test strategy. The test analyst documents the testing requirements in a test plan document by drafting the process of the test roll-out with start and end dates for each test case. The project plan and system test plans are reviewed by the project team members and project manager. The outputs of this phase are a detailed technical design detailed in the SRS, a test plan and project status review.

Researcher: What is a project status review?

Participant 1: The project status review involves assessing the status of the project in terms of timelines, objectives achieved, cost incurred in relation to budget, and review of risks.

Researcher: How long does this phase take?

Participant 1: This phase takes approximately 4 – 8 weeks to complete.

Researcher: Who are the available resources?

Participant 1: The project manager, project team members and end users.

Researcher: What happens next?

Participant 1: The development phase. During the development phase the design is converted into a complete information system. Firstly, the test analyst designs the test cases to match the BRS specification. The test analyst proceeds to create the test data. This involves setting up the variables and fields necessary for testing.

Researcher: How is the test data created?

Participant 1: The test data is manually manufactured by the test analyst by developing scenarios that test the software robustness. These scenarios cover all possible combinations related to the software change. Better explained, once the testers have determined which scenarios need to be tested, they create a customer record that relates to each scenario. Approximately 20 to 30 customer profiles are manufactured. Parallel to creating test cases, CMS and the developer acquire and install their respective system environments (for example INT or QA) and proceed to create and test the databases. This usually involves changing and adding on segments to the existing databases as well as refining programs. The test plan and scripts as well as the project plan is reviewed. The developer proceeds to code the software. The developer and quantitative analyst tests the software changes. Finally, the testing team performs a test readiness review in order to verify whether everything at this stage is ready for the testing phase to commence. The scope, risks and issues are analysed once again. Procurement activities are performed to ensure that the appropriate resources are allocated to the phases that follow. The project plan, code and infrastructure changes, and unit test results are reviewed by the project manager.

Researcher: What are the timelines for this phase?

Participant 1: This phase takes approximately 2 - 6 weeks to complete.

Researcher: Who are the available resources?

Participant 1: The available resources are the test analyst, CMS, the developer, the quantitative analyst and the testing team. The deliverables are the coded solution into SM software, the test pack, the test plan, code scripts, unit test results, code and infrastructure reviews, review of project status, scope, risks and issues. The testing phase commences once the outputs of the development phase have concluded. During the testing phase Hogan firstly conducts performance or load testing. A large number of records which exceeds the number of records in a production environment are executed through the newly developed software. This is conducted in order to verify that the system will be able to handle a realistic amount of records in the production environment. It also assists in verifying that the integrity of the data is not comprised. Reports are produced and analysed. Compatibility testing is performed which is a test for compatibility with other channels, operating systems, old or new versions, or target environments. The outcome is in the form of a test case document. Compatibility testing is followed by integration testing. Integration testing consists of 2 separate phases: The Integration testing phase (INT) and the Quality Assurance testing phase (QA).Functional testing is performed during the Integration testing phase (INT) on the integrated components and system. Functional testing is performed in order to demonstrate that the developed system conforms to the requirements as specified in the Functional Requirements Document. Once positive results have been received, the phase is subsequently approved in order to migrate the system to the Quality Assurance (QA) phase. The deliverables are the test case execution and functional testing. The available resources are the project manager, technical team lead, developer, business analyst, and test analyst.

Researcher: How long does this phase take to complete?

Participant 1: Approximately 4 weeks.

Researcher: What happens during the QA phase?

Participant 1: Quality Assurance testing follows functional testing. During Quality Assurance testing, the test analyst performs regression testing. The available resources are the project manager, the technical team lead, the developer, the business analyst, and the test analyst. The QA phase takes 2 weeks to complete. The test analyst finally retests the defects until the results are satisfactory. Once testing has obtained the desired results, the test manager produces a

test coverage and sanity report of the cycle. The testing team subsequently approves the migration of the system into the production environment.

Researcher: Who are the available resources for this phase?

Participant 1: The project team members, project manager and business unit and the deliverables are the test coverage report, defect report and project status review.

Researcher: What is next?

Participant 1: In order to validate the system, acceptance testing is performed by the relevant business unit with the help of the development team. Once the business unit has accepted the system, the project status is reviewed and formal approval is given to implement the system into the production environment. The outputs of this phase are the test results, the users' approval to implement the system, handoff of the operational technology, and reviews of project status. The implementation phase includes implementation preparation and implementation of the system into a production environment. Once business unit acceptance is received and positive test results are obtained for system and regression testing, Hogan submits the change for implementation by producing the final package to install into the live environment. The data is then migrated from the QA environment to the production environment according to the configuration management details. Subsequently the analytics team and business unit extract a sample of records to verify whether the data is flowing through the correct path of logic. Finally, the system is implemented into a production environment. The system is operated according to standard system operating and management procedures. Once the system is in production, the test analysts performs maintenance testing which is known as Post Implementation Testing (PIT) in order to verify that the environment is stable and that nothing is broken, i.e. that defaults do not appear in the production environment. Maintenance of the system is initiated once the system has been moved to the production environment, the system has been verified and validated, and the project has been signed off. The available resources are the project team members, project manager and business unit. The deliverables are the Post Implementation Review, the configuration management records and the project closure. **Researcher:** Can you provide examples of modifications?

Participant 1: Examples of modifications to the existing system are fixing of defects and enhancements and upgrades of the system such as changing the infrastructure or upgrades to the software. It could also include corrective and emergency fixes and environment changes.

Researcher: What do you perceive to be the problems in the current SDLC?

Participant1: The changing of requirements poses a great risk to project success. It often occurs that that the business unit's requirements change during the life cycle. For example, after development the business unit realises that they need to change their requirements.

Researcher: Why does this happen?

Participant1: The business unit submits requirements that they think will achieve their objectives. Only later in the development do they realise that what they requested is not actually what they want.

Researcher: How could the problems be addressed?

Participant 1: By spending more time analysing the business requirements.

Researcher: What do you perceive are the strengths within the current SDLC?

Participant1: The software development follows a well-defined process by which everyone adheres to.

Interview transcript - Participant 2 (Developer)

Researcher: Describe the area you are involved in

Participant 2: I am the developer responsible for coding software changes and conducting unit testing. I work in the development area.

Researcher: Describe the processes involved this area and their order

Participant 2: I integrate the system by coding the software changes into SM according to the SRS, FRS, test plans and coding and infrastructure standards. I integrate the system incrementally according to the technical design and ensure that each integration step achieves the desired system functionality. I ensure that the system is integrated and interfaced with other systems as required, that all operational systems are compatible with each other, and that the correct versions of and correct components are used for integration. Along with the quantitative analyst I subsequently perform unit testing otherwise known as component testing.

Researcher: Can you describe how unit testing is performed?

Participant 2: During unit testing, I test whether the coded input is resulting in output, but do not test whether the results are correct. Once this has been done, I provide notification to the quantitative analyst that the coded changes in SM have been completed. The quantitative analyst proceeds to verify the code by testing whether the coded changes made to SM are resulting in the correct output.

Researcher: Explain how each process works and describe the inputs and outputs of the different processes

Researcher: Provide detail on resources, departments and timelines involved with each process

Participant 2: Software development is performed by the developer. Unit testing is conducted by the developer and quantitative analyst. Approximately 2 weeks are provided for development and 1 week for unit testing.

Researcher: What do you perceive to be the problems in the current SDLC?

Participant 2: Functional testing performed by the test analysts are highly reliant on the unit testing results.

Researcher: How so?

Participant 2: The test analysts perform functional testing manually. They manually manufacture a few customer profiles and manually manipulate the input fields. The problem is that it is time consuming and the quality of testing is poor as it is not performed on a representative customer data base.

Researcher: How could the problems be addressed?

Participant 2: By automating testing and performing testing on a sample that is representative of the customer data base.

Researcher: What do you perceive are the strengths within the current SDLC?

Participant 2: The system design, system development, and unit testing.

Interview transcript - Participant 3 (Quantitative Analyst)

Researcher: Describe the area you are involved in

Participant 3: I work in the software development area. I am the quantitative analyst responsible for conducting unit testing and assisting the test analysts with integration and system testing.

Researcher: Describe the processes involved this area, their order as well as the inputs and outputs of the different processes

Participant 3: Unit testing is performed by developing test cases.

Researcher: How are test cases developed?

Participant 3: Different scenarios are set up that are related to the change. Mathematical algorithms are used in order to automate unit testing. These algorithms run the test cases through the data obtained from the production environment. Once the code has been verified, the quantitative analyst provides sign-off confirming that unit testing has been completed. Unit testing code is stored in order to be re-used during Post Implementation testing or another change request.

Researcher: What happens then?

Participant 3: The testing phase commences. The test analysts perform functional testing. It involves both positive and negative testing. The test analysts proceed to manually manipulate the relevant input of each manufactured customer record to see if the desired output is obtained.

Researcher: Can you give an example?

Participant 3: For example let's say the request was to decline all customers for a loan that have a monthly net income of less than R10000. The test analyst would then extract a customer record where the customer has a net monthly income of less than R10000 to verify that the application is declined (positive testing) and a customer record where the customer has a net monthly income of more than R10000 to verify that the application is not declined (negative testing). The test analyst thus verifies that the customer is approved or declined correctly. The outcome is in the form of screen shots.

Researcher: Are the test results documented?

Participant 3: The test results are documented and archived for future use.

Researcher: Provide detail on resources, departments and timelines involved with each process

Participant 3: 1 weeks' allowance is given for unit testing. Unit testing is performed by the developer and quantitative analyst. Functional testing takes 3 weeks as the test analysts have to manually manufacture data and perform testing.

Researcher: What do you perceive to be the problems in the current SDLC?

Participant 3: Test analysts do not adequately verify that the system has been tested. They create a few phantom customer profiles on which to perform testing. It is not adequate.

Researcher: Why is it not adequate?

Participant 3: Testing should be performed on a representative sample of customer profiles. This will be more representative of the live environment.

Researcher: How could the problems be addressed?

Participant 3: By extracting live customer data

Researcher: What do you perceive are the strengths within the current SDLC?

Participant 3: Project management is very good. The project managers follow a set out process. Timelines and responsibilities are assigned which makes your own work simpler and easier.

Interview transcript - Participant 4 (Test Analyst)

Researcher: Describe the area you are involved in

Participant 4: I am a test analyst in the HTQA testing team. The HTQA testing team is responsible for testing software changes.

Researcher: What type of testing do you perform?

Participant 4: System testing thus functional and regression testing.

Researcher: Describe the processes involved this area, their order, inputs and outputs

Participant 4: The test analysts firstly set up a test plan according to the business requirements with end dates for each activity. The test plan is reviewed by the project team members.

Researcher: During which phase does this happen?

Participant 4: During the testing phase

Researcher: What happens next?

Participant 4: The test analysts design test cases according to the BRS. This involves creating a set of conditions under which the tester determines whether a project is working according to the user needs. The test analyst then tracks the test coverage by means of a documented test coverage matrix (TCM). A TCM is constructed by creating a checklist which ensures that the functionality of each software unit is checked in all possible combinations so positive as well as negative, and special conditions. The outcome is in the form of a matrix document. This is done to ensure that all probable conditions and cases for a feature to be tested are thought through. It also assists in identifying probable gaps. The test cases and scripts are also reviewed, modified and refined by the testing team. The test analysts then proceed to create the test data. This involves creating the necessary fields required for testing.

Researcher: Can you give an example

Participant 4: For example the business requirement is to decline all customers with a Risk Category code of 3 and above. What happens is that we create the input field “Risk Category code” and manually manipulate it to view if the expected results are obtained.

Researcher: How do you create the test data?

Participant 4: Customer profiles are manually created with the necessary characteristics required for testing. Approximately 20 to 30 customer profiles are created.

Researcher: What happens then?

Participant 4: Software development and unit testing takes place. Once the test results have been approved, the testing phase commences. The testing phase is divided in two phases: Integration phase and the Quality Assurance phase. During the Integration phase Functional testing takes place. It is performed by conducting component integration testing, system integration testing, and finally system testing. Structural testing is performed to complement functional testing. This ensures that testing has been performed thoroughly by testing different scenarios that pass through the different flows in the architectural design. Structural testing is performed by testing the architecture of the system or component in order to determine how thorough the testing up to a certain point has been.

Researcher: What do you mean by architecture?

Participant 4: The architecture refers to the process logic of the credit scoring software. Structural testing compliments functional testing by testing a set of conditions that cover different elements of the architecture. It provides a measure of how thorough the tests have been in order to establish if more tests are needed to gain the necessary coverage of test cases.

Researcher: When is structural testing performed?

Participant 4: Structural testing is performed at all levels of the V-model where it is deemed necessary. For example, it is performed at component level in order to test the code coverage, integration level in order to test the module coverage, system level in order to test the final system coverage, and finally at acceptance level to test the business model coverage.

Researcher: Can you give an example of structural testing?

Participant 4: For example, a customer that has an income of above R10000 per month might pass through different scoring logic compared to a customer that has an income of less than R10000 per month. Test Analysis Reports are produced during this stage. Should defects be discovered during functional testing, the test analyst will resolve the defects with the developer

and retest the defects in order to ensure that the problem has been resolved. The test manager will proceed to produce a defect and test coverage report in order to verify that possible scenarios were not left out. Once testing results are satisfactory, formal approval is given to migrate to the QA phase. This is where regression testing is performed.

Researcher: Can you explain how regression testing is performed?

Participant 4: In the current SDLC a regression test pack is used that consists of one customer profile that is manipulated according to the test cases that cover the change. Only positive testing is performed on the customer profile. For example, if the change was to approve customers with a monthly net income of greater than R10000, a customer profile is selected from the production environment that has a monthly net income of greater than R10000 to verify that the customer is approved. However, tests are not conducted to verify that customers with a net monthly income of less than R10000 are approved. Regression testing is automated. Test execution tools are used in order to execute the tests. A test results document and screen shots are used to review the results. Once regression testing has been approved, approval is given to move the system into the live environment. Once the system has been moved into the live environment, maintenance testing which is known as Post Implementation testing is conducted. Depending on the scope of the changes required, previous life cycle phases are revisited where necessary. A quality assurance report is subsequently produced in order to state that the system is stable and that no defects are occurring.

Researcher: Who performs PIT testing?

Participant 4: Maintenance testing is performed by the testers of the HTQA team.

Researcher: How is maintenance testing peformed?

Participant 4: The testing of changes is in depth and focus on the specific areas where the changes have occurred. Testing is performed by testing the entire integrated system first in order to see the effect on the whole system. Once the results are satisfactory, the specific area where the change was made is then tested. Data from the production environment is extracted and the data is run through the same logic used during unit testing. Production testing is easier than testing during development as live data is available and it is therefore not necessary to go through the process of building test data. Maintenance testing also involves performing regression testing. Regression testing needs to be performed as the software changes made can cause other areas of the system to be affected. Regular testing of changes may not pick up

all possible defects as this only involves testing the area of the software pertaining to the change. Once the system has gone live, i.e. moved to a production environment, monitoring is additionally set up to monitor the system going forward for any defects. This information is continuously analysed and verified to assist with the traceability of defects. Finally, the system is monitored in order to make changes where necessary and reports on the stability of the system are produced.

Researcher: Provide detail on resources, departments and timelines involved with each process

Participant 4: Available resources for testing are the test analysts, testing team, quantitative analysts and developer. Also the project manager. Time allocated for the creating of test cases and data is 2-6 weeks. The INT phase is 4 weeks and the QA phase is 2 weeks

Researcher: What do you perceive to be the problems in the current SDLC?

Participant 4: There is a need to automate testing of the INT phase. The manual process is time consuming. An adequate regression test pack is required that will include customer profiles that are representative of the production environment. 1 customer profile is not adequate.

Researcher: How could the problems be addressed?

Participant 4: Automation of system testing and using live customer data.

Researcher: What do you perceive are the strengths within the current SDLC?

Participant 4: Each activity is reviewed and approved before proceeding to the next

Interview transcript - Participant 5 (Systems Analyst)

Researcher: Describe the area you are involved in

Participant 5: I form part of the development team responsible for delivering the software to meet business' requirements.

Researcher: Can you elaborate on your role?

Participant 5: I am the systems analyst responsible for designing the software system according to the business requirements.

Researcher: Describe the processes involved this area with their respective inputs and outputs

The BRS is sent out to the respective resources. An impact assessment is scheduled and the various resources come to an agreement on the requirements and risks involved. Once agreement is obtained and it is confirmed that a FRS is required, the FRS is developed.

Researcher: Who develops the FRS?

Participant 5: The business analyst

Researcher: What happens once the FRS is developed?

Participant 5: Development activities are planned and the development activities are translated into requirements. A preliminary system design is developed and translated into an SRS.

Researcher: How is the system designed?

Participant 5: The system is designed according to the system functional architecture as detailed in the BRS and FRS. The design involves detailing the function and performance of the system, developing the required architecture and defining how the system should be verified. Firstly, the various elements and subsystems are defined. The design includes the design of the interfaces between different elements, subsystems and systems. For example, the design includes investigating how to send the data to SM and other business units as well as investigating which information is required from the various business units in order to develop the system as per the requirements. The system is analysed by identifying all interacting components and systems involved, considering all possible solutions that would address the problem and determining the system boundary. Alternative design solutions are evaluated

based on aspects such as performance, schedule, cost and risk. The best design solution is selected by developing different models and prototypes of the different solutions and running trials in order to identify the solution that produces the best outcome.

Researcher: How is the design verified?

Participant 5: It is ensured that the design is simple and understandable by the developers and end users. Input and review from other resources and the end users are obtained.

Researcher: Provide detail on resources, departments and timelines involved with each process

Participant 5: The design process takes approximately 2 weeks. The systems analyst, testing team, project manager, developer and business unit is involved in order to ensure that the requirements are correct and the design is according to the requirements.

Researcher: What do you perceive to be the problems in the current SDLC?

Participant 5: Often requirements change. You will have designed the system and the business unit decides that they want to change the requirements.

Researcher: How could the problems be addressed?

Participant 5: I think the business unit should be more involved in the design of the system. I think the project should be rescheduled when the requirements change significantly. Often the same project timelines are kept which places everyone under a lot of pressure.

Researcher: What do you perceive are the strengths within the current SDLC?

Participant 5: That requirements are well documented. If requirements are not documented, mistakes are likely to occur as assumptions are made.

Researcher: I just have a few additional questions to ensure that I am understanding this correctly. Functional testing involves testing only the functions of the system and is performed by the test analysts?

Participant 5: Yes, just the functions.

Researcher: Test cases are developed according to the BRS? And it is done by developing different scenarios of the change?

Participant 5: Yes that is correct. They test that for each case the change is tested.

Researcher: Integration is performed by the developer during software coding?

Participant 5: Yes the developer integrates the system and does unit testing. The developer codes according to the SRS. He can use the FRS if required. The FRS provides extra clarity on what the business wants related to the system.

Researcher: Do testers just perform functional and regression testing? Where does non-functional testing fit in?

Participant 5: They perform all system testing during the INT phase. This consists of functional, regression and structural testing. Performance testing is also performed.

Researcher: How is the SRS and FRS developed?

Participant 5: The FRS contains the functional requirements of the system for example if you click Q it writes Q. The SRS contains the system requirements for example what the system must look like and where the function will be placed in the system. It tells you where in the system the functional changes are going to happen and how. For example the Q button must be left of the W button. Testers perform integration testing during INT phase. That is known as functional testing. The testers test whether the functionality is doing what it is supposed to do. Testers also perform structural testing which is in the SRS.

Researcher: And then last question. The test plans are created from the FRS or BRS?

Participant 5: The testers will look at the BRS and FRS and create test plans from FRS. The FRS contains the integrated picture of the system.

Interview transcript - Participant 6 (Business Analyst)

Researcher: Describe the area you are involved in

Participant 6: I am involved in the development area responsible for implementing change requests.

Researcher: What is your role and responsibilities?

Participant 6: I am the business analyst responsible for developing the business requirements into a functional specification that can be used to design and develop the system.

Researcher: Describe the processes involved in your area?

Participant 6: My role stretches from the requirements analysis phase through to the design phase. During the requirements analysis phase the BRS is submitted by the business unit. The BRS contains the business requirements. The BRS is assessed by all the resources involved. During the impact analysis meeting the current environment is assessed and performance, operational, functional, and interface requirements are formulated, quantified, and refined according to the business' needs. All possible solutions are considered so that the best solution can be reached. The best requirements and solution are selected by comparing it to the business' requirements. It is ensured that the requirements are clear, unambiguous, accurate and complete. The requirements include details on how the problem should be solved (how the system should be designed), how well the problem should be solved and how the system should be verified and validated. Security requirements are developed in order to conform to the Information Security Standards (ISS). Design constraints are defined and clarified. Finally, the requirements are prioritised and categorised in order to ensure that the critical requirements are met first should all the requirements not be able to be completed within schedule. The requirements are subsequently organised from high level to low level requirements in an understandable and traceable manner.

Researcher: How is the FRS developed?

Participant 6: The FRS is developed by allocating the functional and non-functional requirements to the different system elements. The system's functions are decomposed and requirements are developed for each function. It is important that the FRS is detailed enough in

order to construct the system accurately and that it is designed in an orderly manner according to customer needs and technical capabilities.

Researcher: What happens next?

Participant 6: The FRS is approved by the project team. The system is subsequently developed.

Researcher: Provide detail on resources, departments and timelines involved with each process

Participant 6: The requirements analysis phase takes approximately 2-6 weeks depending on scope and rework. The entire project team, developers, systems and business analysts are involved.

Researcher: What do you perceive the problems in the current SDLC to be?

Participant 6: I find that the business unit can often not decide on their requirements. They keep on changing their requirements throughout the development process.

Researcher: How could the problems be addressed?

Participant 6: I guess more detailed analysis of the requirements by the business unit

Researcher: What do you perceive are the strengths within the current SDLC?

Participant 6: A well defined development process by which all the resources can adhere to. Knowing ones roles and responsibilities with timeliness assigned keeps everyone committed to reaching the required implementation dates.

Researcher: How are timelines assigned?

Participant 6: There is a time schedule for all the phases of the SDLC which needs to be adhered to. Task timelines are adjusted as necessary throughout the project.

Interview transcript - Participant 7 (Test Analyst)

Researcher: Describe the area you are involved in and your roles and responsibilities?

Participant 7: I am part of the HTQA testing team. We are responsible for testing the system for any defects and ensuring that the system is aligned with business requirements. I am responsible for conducting performance, load and compatibility testing.

Researcher: Which phase of the life cycle does this involve?

Participant 7: The testing phase. I am also responsible for maintenance testing which is performed during the implementation and maintenance phase.

Researcher: Describe the processes involved this area along with their order, inputs and outputs

Participant 7: The testing phase commences by conducting performance and compatibility testing. Performance testing measures the performance of the software by performing timing tests such as response times. It monitors the behaviour of the system by logging the number of transactions and response times of these transactions. Reports are produced based on the logs and graphs of these load versus response times. Performance testing tools such as a user interface or test harness are used to generate realistic loads on the system, database, or environment in order to determine the behavior of the system. Load testing is performed by testing a volume of records that is representative of the production environment in order to verify that the system will be able to handle the volume of records in the actual production environment. During load testing, aspects such as processing throughput and the number of connected terminals are tested. Storage testing is performed by testing whether the objectives for storing are being met. There can be various objectives for storage, for example, a limited amount of space on the server may be used for certain operations or a limited amount of customer data may be kept in the data warehouse for certain customer profiles. Storage testing is usually applicable to embedded software, for example software embedded into the various data warehouses. Reliability testing is performed in order to determine how reliable the system is. In other words whether the system has a low probability of failure. Stress testing methods are used to conduct reliability testing. Stress testing is performed by testing whether a volume of records that is beyond that found in the production environment can be handled by the system. This serves as a safety and risk measure to ensure that the system will be able to handle additional number of records added to the system in a production environment. Once these

tests have been concluded, the results are reviewed and sign-off is given to proceed with INT testing.

Researcher: And maintenance testing?

Participant 7: Once the system has been moved to a production environment, maintenance and maintenance testing is conducted. Maintenance involves assigning resources to ensure that the different components and system as a whole is functioning. Maintenance testing is performed in order to ensure that defects do not occur in the production environment. Maintenance testing is also performed when changes to the existing system in the production environment are made. These changes can be due to modifications, migration to new platforms or retirement of the system. Maintenance testing is conducted on a daily basis in order to ensure that the system is still functioning according to the requirements. Improvements and adjustments are continually made where necessary in order to ensure that the system keeps on working as required. Aspects of sample changes made to the system are tested. It is performed in order to ensure that the system will be able to operate in the different configurations. Testing is performed in a representative set of configurations of the production environment. Configurations refer to the versions of the software. For example, configurations used by the business unit could be the different versions of the credit scoring system. For example, one version of the credit scoring system could be used for monitoring and another version could be used for applications. Portability and interoperability testing also involves testing the interconnected data paths in the system. This is especially important when a part of the system has been upgraded as this can now result in conflict with other parts of the system that have not been changed.

Researcher: Provide detail on resources, departments and timelines involved with each process

Participant 7: The HTQA test analysts are responsible for conducting testing during the testing phase. However the results are reviewed by the project manager, project team and business unit. The entire testing process takes approximately 4-6 weeks however performance and compatibility testing takes approximately 1 week.

Researcher: What do you perceive to be the problems in the current SDLC?

Participant 7: Poor quality of testing. It often occurs that defects are not picked up when it should have been picked up. More rework is therefore required.

Researcher: During performance and compatibility testing?

Participant 7: No, testing during the INT and QA phases. I have tested the functionality of the system before and I don't think it is the most optimal process.

Researcher: How so?

Participant 7: The process is not automated which is very time consuming.

Researcher: How could the problems be addressed?

Participant 7: Automation of all testing processes. Actually all processes if possible.

Researcher: What do you perceive are the strengths within the current SDLC?

Participant 7: Accurate development of specifications and design and development according to the specifications. The quality of verification however is a problem.

Interview transcript - Participant 8 (Test Manager)

Researcher: Describe the area you are involved in along with your role and responsibilities

Participant 8: I am the test manager that oversees system testing. I work in the HTQA department which is responsible for testing software changes made to the credit scoring logic.

Researcher: Describe the processes involved in this area and their order. In other words, how is testing in your area performed.

Participant 8: Firstly performance and compatibility testing is performed in order to see if the system will be able to operate in the production environment and whether it will be able to be integrated with other system. System testing is subsequently performed. Firstly functional testing and structural testing is performed. Functional testing tests whether the system is functioning according the requirements. Structural testing compliments functional testing by testing the entire architectural flows of logic. Functional and structural testing form part of the INT or Integration phase.

Researcher: You mean it is where integration testing takes place?

Participant 8: Correct, the system is tested in increments according to the integration order during system development. Once the results prove to be satisfactory, the system is moved to the Quality Assurance or QA phase. This is where regression testing takes place. Regression testing tests the system by ensuring that the software change has not caused unwanted side effects or defects.

Researcher: What happens if defects are detected?

Participant 8: If any defects are picked up during regression testing, the defects are resolved with the developer. The test analyst compiles a defects list which is a matrix of defects, showing the description of the defect, the date the defect was identified and the person the defect is assigned to.

Researcher: How is regression testing performed?

Participant 8: Regression testing is performed by executing the same test cases or scenarios as was used during functional testing. Technically regression testing should be performed on

real life customer data so that the live system can be accurately represented. Currently only 1 customer profile is created on which the test cases are executed. This will have to be looked at.

Researcher: Regression testing is automated, right?

Participant 8: Correct, it's automated.

Researcher: What happens once regression testing has proved successful?

Participant 8: The system is moved to the live or production environment. Post Implementation testing is conducted by the HTQA team in order to ensure that defect do not occur in the live system.

Researcher: Provide detail on resources, departments and timelines involved with each process

Participant 8: The testing phase takes 4-6 weeks to complete. This can be reduced by improving the accuracy of testing.

Researcher: What do you perceive to be the problems in the current SDLC?

Participant 8: As stated, the quality of regression testing. The same problem exists with functional testing where real life data is not used but instead customer profiles are manufactured. Only a few customer profiles are created. This does not represent the live environment accurately. Additionally due to the lack of automated testing during the INT phase. Approximately 6 testers are required. It has been worked out that the capacity utilisation is over 300%. Unnecessary time is spent on testing.

Researcher: How could the problems be addressed?

Participant 8: This is what I suggest. We extract a sample of approximately 10-15% of customer account data from the production environment for test cases. Data can be extracted from the different data warehouses in order to obtain the necessary field information. Obviously filtering requirements will have to be set up in order to have a variety in spread and selection of data. This will also assist in achieving the necessary test coverage as well as testing a wider variety of scenarios. As testing is performed in an uncontrolled environment which means that the testers will be able to view actual customer data, the data will need to be desensitised and depersonalised by breaking the link from the production profile to the test data. In other words remove personal information such as names from the test profile. The customer account data

can then be re-matched by means of algorithms when it needs to be sent to the Credit Bureau to obtain the customers' credit bureau information. Once again the same logic can be used to depersonalise the records once it is receive back from the credit bureau. I suggest automating functional testing by running mathematical algorithms through the entire sample as is done with unit testing.

Researcher: What do you perceive are the strengths within the current SDLC?

Participant 8: The expertise of our resources. Our resources carry vast amounts of knowledge and expertise with them. The problem is when they leave. I think experience and knowledge should perhaps be documented.

Interview transcript - Participant 9 (Business Unit Analyst)

Researcher: Describe the area you are involved in and your role and responsibilities?

Participant 9: I am an analyst from one of the business units in the bank. I am involved throughout the SDLC phases. I represent the business unit and my responsibility is to ensure that the business unit's requirements are met throughout the entire software development process.

Researcher: Describe the processes involved this area, their order, inputs and outputs

Participant 9: Firstly the business unit will analyse their own objectives and develop a requirements specification known as a BRS that details the business unit's objectives. The BRS is subsequently sent to the project manager who will distribute it to the project team. In impact analysis meeting is scheduled where factors such as technical size, technical complexity, technical quality, technical value, project duration, project cost and risk are taken into account and discussed. The resources from the various disciplines and divisions in the SDLC as well as the business unit representative provide comment, raise concerns, and give their opinions on how to proceed. The outcome of these meetings are to ensure that all involved understand the entire system, the trade-offs, the derived problem definition, and that agreement is obtained so that everyone including the stakeholders will accept the solution. Risk is considered during the selection of users' objectives and design. Knowledge regarding the technical aspects of the system is shared between the project team resources and end users. Successes and similarities of previous projects and lessons learnt by the different resources are discussed and taken into account in order to reduce the probability of defects and improve quality and schedule. Once agreement on the requirements is obtained by all involved the project is sized and scheduled. The design, development and testing phase follows. The involvement of the business unit during these phases is to review the design, code and test results. Currently the design and code is reviewed. User acceptance testing is conducted just before implementing the system. The objective of user acceptance testing is to allow the business unit to gain confidence in any aspect of the system, determine whether the system meets their needs, and determine whether the system is ready to implement. It is performed by the end users instead of the technical staff because it is often difficult for people from a strong technical background to know when the users will experience the system as being user-friendly as they might perceive a more complex system as quite simplistic due to their computer-literate skills.

Researcher: How is user acceptance testing performed?

Participant 9: The business unit tests a random selection of data in order to determine whether the business requirements are met. Any test environment and any test is performed that they deem necessary. The business unit designs their tests at the same time as developing the Business Requirement Specification. This helps the business unit to ensure that their requirements are logical and realistic. Otherwise, the business unit may realize only at the end of the development process that their requirements are not correct. This will results in requesting changes very late in the SDLC that can cause schedule, cost, and quality to be compromised. The business unit develops the specification for acceptance testing at the beginning of the project. Technically, not many defects should be present at this phase of testing as most of the defects should have been resolved during functional and regression testing. Acceptance testing occurs at various phases. For example it is performed after component testing to ensure that the components are usable and it is performed after system testing or system integration testing to ensure that the functionality of the system is aligned with the user specifications. Typical defects found during acceptance testing are mismatches with business needs and misunderstandings of business processes.

Researcher: Provide detail on resources, departments and timelines involved with each process.

Participant 9: Timelines are according to the project schedule handed out by the project manager at the start of the project. Although roles and responsibilities are assigned to individuals, all resources are encouraged to participate throughout the process.

Researcher: What do you perceive to be the problems in the current SDLC?

Participant 9: Inadequate testing. Problems are usually picked up during user acceptance testing that should have been picked up earlier in the life cycle.

Researcher: How could the problems be addressed?

Participant 9: By investigating the testing process.

Researcher: What do you perceive are the strengths within the current SDLC?

Participant 9: Good communication between the resources and the project manager.

Interview transcript - Participant 10 (Project Manager)

Researcher: Describe the area you are involved in along with your role and responsibilities

Participant 10: I am a project manager responsible for ensuring that the system is designed, developed and implemented according to the business requirements in the required time. Project managers are involved throughout the entire SDLC. The project manager estimates the time, cost, scope and resources according to the specifications and risk. Throughout the SDLC, the project manager is responsible for planning the project according to the desired timelines and budget, developing a work breakdown for all resources, ensuring that the duties and responsibilities of all involved are executed according to the schedule, ensuring that the different elements, components and interfaces are combined and tested in an efficient way in order to meet the stakeholders' requirements, and keeping track of and documenting requirements changes. The project manager therefore needs to have a thorough understanding of the SDLC, its complexities, weaknesses, strengths, interdependencies, processes, and the environment in which the system will operate, and assigning resources accordingly. The project manager also needs to ensure that stakeholders are kept involved throughout the entire process in order to ensure that the system is being developed according to their needs. The project manager identifies, mitigates and resolves risks throughout the SDLC.

Researcher: Describe the processes involved this area, their order, inputs and outputs

Participant 10: System development and testing takes place according to a development model. The current development process is based upon a V-model design.

Researcher: How is the system developed and tested according to the V model?

Participant10: The lowest level in the V-model is component development and testing.

Researcher: What is a component?

Participant 10: A component is a small piece of software that is one of the building blocks of the software. A piece of software could be a few lines of code, a small program or database modules. The component is thus the lowest level item that is testable and is tested in isolation if possible in order to ensure that it is tested in detail.

Researcher: How is component testing conducted?

Participant 10: The purpose of component testing is thus to test the detail of the coded software. Once the software component has been coded by the developer, the component's functionality, structure and interfaces are tested in the development environment. If defects are detected, they are resolved as soon as they are found by recoding the software. The defects are recorded as this might assist with detecting and solving defects in other parts of the system. Once the component has been coded by the developer, component testing is performed by the quantitative analyst and the developer. The developer is able to find defects and their causes quickly as he or she understands the logical flow of the code. If component testing were to be performed by someone other than the developer, this could result in more time spent on testing as it would take another resource much longer to find the cause of the defects and each defect would have to be documented, reported and explained to the developer. The developer would then have to review and analyze the reports and fix them. Before component testing is performed, the tests are designed by the developer based on the component specifications and code. Both functional and structural test cases are designed depending on the risks, importance and complexity involved. The next level of the V-model is integration development and testing.

Researcher: How is this performed?

Participant 10: Integration development and testing is performed by combining components that have already been tested into larger assemblies so that testing can be performed on the newly formed assembly. Testing on the assembled part is performed by testing that the components correctly function together by looking at the interfaces between them. This assists in detecting defects that weren't previously detected when the components were tested in isolation. It can occur that when components are combined, certain aspects of one component could result in functional failure of another component. For example, although interfaces were tested during component testing in order to ensure that communication occurred from one side, integration testing could pick up that the communication between the different interfaces aren't working from all sides. The objective of integration testing is thus to test the interactions of the integrated software part. Integration testing is based upon the software system design, architecture, and test cases. During integration testing not only is the functionality of the interfaces tested, but non-functional quality characteristics are also tested such as performance and structure. It is important to pick up performance and structure issues earlier in the process as performance and structural changes may occur as more and more parts are assembled. Resources are planned before integration testing commences. Integration development and testing is performed at multiple levels of the V-model such as at component development and

integration testing and system development and integration testing. Component integration testing involves testing the interfaces and interactions between components that have been combined. It is performed just after component testing has been performed by the HTQA testing team. System integration testing involves testing the interfaces and interactions between smaller parts of the system, including hardware and software, that have been combined to form the final bigger meta-system. This is performed just after sub-system testing by the HTQA testing team. Integrating systems and components can be a complex task when many different components, systems, interfaces and areas of the organisation are involved. Therefore integration is done in increments or steps based on the architectural design of the system. Components are combined into an assembled part and tested. This ensures that the basic parts are integrated and tested first before continuing to more complex integration. This is followed by combining the different tested assembled parts and testing it as a new assembled part. This continues until the entire system has been integrated and tested. Therefore a bigger part of the system is assembled only when the smaller parts have proved to be working and are trusted. This process is known as incremental integration. Incremental integration is performed in order to discover and fix defects easily and at the earliest opportunity but also offers faster and easier recovery if defects are detected. In order to ensure that integration testing is performed as efficiently as possible, the testers of the HTQA team are involved as early as possible in the development of the software. Before integration testing commences, the test analyst determines the integration strategy by determining how many components need to combined in each step and the sequence of combining the components. This is done by looking at the architecture, functional tasks, and processing sequence. The test analysts of the HTQA team prepare test cases according to the test strategy, prepare the test data, test according to the requirements and design, compare test results and manage defects. The testing team has a thorough understanding of the software architecture which assists them in performing integration testing. The testers first plan the order in which the tests will be integrated. The order in which the actual components or system is integrated is therefore based on the order of integration testing. This results in more efficient testing as well as reduced time spent on development.

Researcher: So system testing is performed at all levels?

Participant 10: System development and testing is performed at all stages of the V-model which combines and tests the system as a whole. System testing is performed by the HTQA testers on the entire system in a realistic environment.

Researcher: How is system testing performed?

Participant 10: System testing involves functional, non-functional and structural testing. Tests are based upon system and functional specifications. Firstly, functional testing is carried out in order to ensure that the system is functioning according to the specification. Non-functional and structural tests are subsequently performed on the system to ensure testing thoroughness. The environment in which system testing is carried out is realistic and representative of the production environment in order to ensure that the same factors are taken into account and therefore environmental defects are avoided. Functional testing tests the behaviour of the software in order to determine whether it is functioning according to the requirements detailed in the functional specification that specifies exactly what the system does and how. Test design is based upon these requirements. The requirements as well as the tests are subsequently prioritized based on risk. This ensures that the most important and most critical tests are performed. Non-functional testing tests how the system is performing according to the schedule. Quantifiable units of measurement are defined in order to determine exactly how well the system is performing. Testing is always performed according to requirements specifications.

Researcher: Can you provide examples of non-functional testing?

Participant 10: Examples of non-functional testing performed include performance testing, load testing, stress testing, maintainability testing, reliability testing, portability testing and usability testing.

Researcher: What happens after the system has been tested?

Participant 10: After system testing has been performed, acceptance testing is performed. Acceptance testing is the final stage of validation. User acceptance testing is mainly performed at the end of the development process when the entire system has been integrated. Testing is performed by the final users of the system in order for them to gain confidence in the system and to determine whether the system is ready to be transferred to the production environment. It is therefore the responsibility of the users to ensure that the system is working according to their initial requirements. After user acceptance testing has been performed and the business unit has found that the system validates, sign-off is given by the business unit in the form of a mail stating that the results proved to be satisfactory.

Researcher: What do you perceive to be the problems in the current SDLC?

Participant 10: Lack of appropriate documentation on past projects and the current project. It often occurs that previous steps need to be revisited and one has to investigate what happened. Documentation is not available in all instances.

Researcher: How could the problems be addressed?

Participant 10: This needs to be incorporated into the standard software development process.

Researcher: What do you perceive are the strengths within the current SDLC?

Participant 10: Testing is performed throughout the SDLC. Each development activity has a corresponding test activity. Each step in the SDLC is therefore verified and validated. This prevents picking up problems later in the life cycle when it is more difficult and expensive to fix.