

Software Developer Competency Framework

by

DAVID MUHANGWA MINANI

Thesis submitted in fulfilment of the requirements for the degree

Master of Technology: Information Technology

in the Faculty of Informatics and Design

at the Cape Peninsula University of Technology

Supervisor: Mr Temuso Makhurane

Co-supervisor: Prof Retha de la Harpe

Cape Town

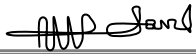
November 2013

CPUT copyright information

The dissertation/thesis may not be published either in part (in scholarly, scientific or technical journals), or as a whole (as a monograph), unless permission has been obtained from the University

DECLARATION

I, David Muhangwa Minani, declare that the contents of this thesis represent my own unaided work, and that the thesis has not previously been submitted for academic examination towards any qualification. Furthermore, it represents my own opinions and not necessarily those of the Cape Peninsula University of Technology.



Signed**07 November 2013**

Date

KEY WORDS

Competency Framework
Computer Programmer
Computer Programming
Developer
Information Technology
Programmer
Software
Software Architect
Software Architecture
Software Developer
Software Development
Software Engineer
Software Engineering
Software Framework
Software Programmer
Software Skills
Software Technology
Software Tools
System Developer
Systems Architect
Systems Consultant
Systems Designer
Web Designer
Web Developer
Webmaster

ABSTRACT

The application of software systems in business organizations continue to increase as the Internet technology grows. Business processes that previously required manual interventions are becoming automated using software systems. The use of software systems is fundamental to electronic processing of business transactions. More business organizations, large and small, are utilizing information technology in order to have competitive advantage in the business arena. Software is ubiquitous. Among areas where software plays core roles are e-Banking where software systems are used to process banking transactions, e-Health where software systems are used to facilitate activities in the health sector, e-Commerce where software systems are used to facilitate online business transactions, e-Government where software system are used to facilitate government activities and e-Learning where software systems are used to facilitate the teaching and learning process.

Nevertheless, the large number of failing software projects and the increase in software security problems coupled with shortage of skilled software developers are still major obstacles in the software development industry. Among others, the solution can be achieved by improving the competency of software developers so that software systems developed are of good quality, safe, robust, and support business objectives. Software companies and business organizations stand a big chance to increase their return on investment (ROI), if competencies of software developers are improved. A software developer plays critical roles in software development projects. A software developer, however, requires specific skills and knowledge in order to develop software systems that solve problems and deliver solutions.

This research is about competencies of software developers. The research focuses on software development activities performed by software companies and business organizations within the Western Cape Province. The unit of analysis is software developers. Data pertaining to tasks performed by software developers, tools used by software developers and skills required were collected, examined and analysed. The objective of the research is to develop a competency framework for software developers. It can be used by institutions and the industry to provide better education. Most importantly, the industry will have access to competent software developers who can perform their job well. As justified in this research, knowledge of a competency framework for software developers is extremely essential.

ACKNOWLEDGEMENTS

I would like to thank everyone who has encouraged or helped me along the way. Without you, my journey of research project to produce this master's thesis would not have been possible.

A special thanks goes out to:

- Prof Retha de la Harpe for encouragement, guidance, advice and support.
- Mr Temuso Makhurane for guidance and advice.
- Prof Richard T. Turley of Colorado State University in USA for advice.
- Prof James Bieman of Colorado State University in USA for advice.
- Mr Raymond Bernardo for promoting the research.
- Dr Andre de la Harpe for advice.
- Mr John Nongalaza for encouragement and advice.
- Mr Swedi Ramadhani for encouragement.
- Mr Roger Norton of Silicon Cape for promoting awareness of the research.
- Ms Jenny McKinnell of Cape Information Technology Initiative (CITI) for promoting awareness of the research.
- Ms Bridgetti Lim Banda for promoting awareness of the research.
- Ms Corrie Uys for advice during statistics data analysis.
- Dr Stuart Warden for initial contribution towards this research.
- My beloved family: parents, brothers and sisters for motivation:
Mr David Tibanywana, Ms Christina Kamulenzi, Mr Muganyizi,
Ms Gladness, Ms Kokubelwa, Ms Kemilembe, Mr Innocent and Mr Allen for
motivation, support and encouragement.
- CPUT RISC Library staff for their support.
- Survey Gizmo team for online support towards data collection and analysis.
- All software developers who voluntarily availed themselves for this research.
- All friends and colleagues who in one way or the other contributed towards the success of this research project.
- Lastly but not the least, the Almighty God for blessings and strength.

The financial assistance of the Cape Peninsula University of Technology towards this research is acknowledged. Opinions expressed in this thesis and the conclusions arrived at, are those of the author, and are not necessarily to be attributed to the Cape Peninsula University of Technology.

DEDICATION

To the memory of my wonderful mother, Christina Kamulenzi Tibanywana;

Mama, I cannot thank you enough, I love you and I miss you dearly;

To the memory of my beloved grandmother Kamarole Tibanywana, and

my beloved grandfather Julius Tibanywana;

I love you and I miss you.

You all will indeed be missed.

May the spirit of our Lord, The Almighty God be with you forever.

Rest In Peace.

TABLE OF CONTENTS

DECLARATION.....	ii
KEY WORDS	iii
ABSTRACT	iv
ACKNOWLEDGEMENTS.....	v
DEDICATION	vi
LIST OF FIGURES.....	xii
LIST OF TABLES	xiii
APPENDICES	xiii
GLOSSARY	xiv
CHAPTER ONE: RESEARCH INTRODUCTION.....	1
1.1 Introduction.....	1
1.2 Background to the research problem	2
1.3 Statement of the research problem	8
1.4 Research question, sub-question and objectives	9
1.5 Literature survey	9
1.5.1 The environment of a software developer	10
1.5.1.1 Inputs.....	10
1.5.1.2 Output.....	11
1.5.1.3 Hardware	11
1.5.1.4 Software	11
1.5.1.5 Data.....	12
1.5.1.6 Business procedure	12
1.5.1.7 People	12
1.5.2 Software Development Life Cycle	13
1.5.2.1 Phase 1: Requirement analysis	14
1.5.2.2 Phase 2: System and software design	15
1.5.2.3 Phase 3: Implementation and unit testing	15
1.5.2.4 Phase 4: Integration and system testing	16
1.5.2.5 Phase 5: Operation and maintenance.....	16
1.5.3 Challenges of a software developer	16
1.5.4 Software developer and software reuse	17
1.5.5 Software project management	17
1.6 Research design.....	18
1.7 Delineation of the research	20
1.8 Contribution of the research.....	20

1.9	Chapter summary	22
1.10	Thesis structure	22
1.11	Conclusion	24
CHAPTER TWO: LITERATURE REVIEW		25
2.1	Introduction	25
2.2	General situation of software projects	25
2.2.2	Software project characteristics	26
2.2.3	Complexity of software projects	28
2.2.4	Project management of software projects	30
2.3	Recent studies on competencies	31
2.4	Software development process	34
2.4.1	Requirements analysis	35
2.4.1.1	Types of software requirements	37
2.4.1.2	Requirements life cycle	39
2.4.1.3	Techniques used during requirements analysis	40
2.4.1.4	Skills required during requirement analysis	41
2.4.2	System and software design	41
2.4.2.1	Advantages of software architecture	42
2.4.2.2	Categories of design	43
2.4.2.3	Service oriented architecture	45
2.4.3	Implementation and software testing	56
2.4.3	Integration and system testing	58
2.4.4	Operation and maintenance	60
2.5	Chapter summary	63
2.6	Conclusion	63
CHAPTER THREE: RESEARCH METHODOLOGY		65
3.1	Introduction	65
3.2	Categories of research paradigm	65
3.3	Research methodology	68
3.4	Typical research techniques used in requirement analysis	69
3.5	Research design for this study	71
3.5.1	Survey questionnaire	72
3.5.2	Interviews	75
3.6	Research population and sampling	76
3.7	Reliability and validity of research methodology	77
3.8	Ethics, Consent and Confidentiality	79
3.9	Chapter Summary	79

3.10	Conclusion.....	79
	CHAPTER FOUR: DATA ANALYSIS AND PRESENTATION.....	80
4.1	Introduction.....	80
4.2	Quantitative data analysis.....	80
4.2.1	Profile of software developer.....	81
4.2.1.1	Gender of respondents.....	82
4.2.1.2	Age of respondents.....	82
4.2.1.3	Race of respondents.....	83
4.2.1.4	Citizenship of software developer respondents.....	84
4.2.1.5	Work experience of software developer respondents.....	85
4.2.1.6	Time taken to become self-dependent software developer.....	86
4.2.1.7	Software development certification status.....	87
4.2.1.8	Education background of respondents.....	88
4.2.1.9	Current position of respondents.....	89
4.2.1.10	Province of respondents.....	90
4.2.2	Software development technologies.....	91
4.2.2.1	Software development environment tools.....	91
4.2.2.2	Programming technologies.....	93
4.2.2.3	Database Management Systems.....	95
4.2.2.4	Code version control tools.....	97
4.2.2.5	Application servers.....	98
4.2.2.6	Software installation environment.....	99
4.2.3	Tasks performed by software developers.....	100
4.2.4	Skills of software developers.....	101
4.3	Qualitative data analysis.....	105
4.3.1	Open-ended question (Other skills of software developers).....	107
4.3.2	Data analysis of interviews.....	112
4.3.2.1	First Company.....	114
4.3.2.2	Second Company.....	117
4.3.2.3	Third Company.....	119
4.3.2.4	Fourth Company.....	124
4.3.2.5	Fifth Company.....	126
4.3.2.6	Sixth Company.....	127
4.4	Chapter Summary.....	129
4.5	Conclusion.....	129
	CHAPTER FIVE: RESEARCH DISCUSSION.....	130
5.1	Introduction.....	130

5.2	Discussion on results of quantitative data analysis	130
5.2.1	Profile of software developers	130
5.2.1.1	Gender of respondents	130
5.2.1.2	Age of respondents.....	131
5.2.1.3	Race of respondents.....	131
5.2.1.4	Citizenship of respondents.....	132
5.2.1.5	Work experience of respondents.....	133
5.2.1.6	Time taken to become self-dependent software developer	133
5.2.1.7	Software development certification status	133
5.2.1.8	Education background of respondents	134
5.2.1.9	Current position of respondents	134
5.2.1.10	Province of respondents	134
5.2.2	Software development technologies	135
5.2.2.1	Software development environment (IDE) tools	135
5.2.2.2	Programming technologies	135
5.2.2.3	Database Management System (DBMS)	135
5.2.2.4	Code version control tools.....	136
5.2.2.5	Application servers.....	136
5.2.2.6	Software installation environment	136
5.2.3	Tasks performed by software developers.....	137
5.3	Discussion on results of qualitative data analysis.....	137
5.3.1	Skills of software developers (Open-ended question of questionnaires).	138
5.3.2	Discussion on interviews.....	138
5.3.2.1	Major skills of a software developer	139
5.3.2.2	Technical ability of new software developers	139
5.3.2.3	Skills regarding software development for mobile devices	140
5.3.2.4	Software applications developed by most of companies	140
5.4	Chapter Summary.....	141
5.5	Conclusion.....	141
CHAPTER SIX: RECOMMENDATIONS AND CONCLUSION.....		142
6.1	Introduction.....	142
6.2	Research summary.....	142
6.3	Recommended SDCF revealed and explained	144
6.4	Recommendations.....	149
6.5	Message to institutions and the industry	154
6.6	Limitations of the study	157
6.7	Future studies.....	158

6.8	Conclusion	158
	LIST OF REFERENCES	161
	Appendix A: Questionnaire	169
	Appendix B: Introductory letter to the industry	177
	Appendix C: Letter to CPUT internship coordinators	178
	Appendix D: Email to software developers	179
	Appendix E: List of companies requested to participate in this research.....	180

LIST OF FIGURES

Figure 1.1: Importance of Competency Framework for Software Developers	5
Figure 1.2: Components of information systems	10
Figure 1.3: The waterfall model	14
Figure 1.4: Thesis structure	23
Figure 2.1: Requirements in the V-model	38
Figure 2.2: Requirements engineering process	39
Figure 2.3: The three-tier architecture	45
Figure 2.4: Four layers of web service architecture	49
Figure 2.5: Web service objects and operations	51
Figure 2.6: V-model showing test plans	59
Figure 2.7: Software installation strategies	62
Figure 3.1: Four paradigms of social research	66
Figure 3. 2: Competencies of software developers under investigation	72
Figure 4.1: Response status of questionnaire	81
Figure 4.2: Gender of respondents	82
Figure 4.3: Age in years	83
Figure 4.4: Race of respondents	84
Figure 4.5: Citizenship of respondents	85
Figure 4.6: Years of work experience	86
Figure 4.7: Time taken to become self-dependent	87
Figure 4.8: Software development certification status	88
Figure 4.9: Education background of respondents	89
Figure 4.10: Current position of software developer	90
Figure 4.11: Province of respondents	91
Figure 4.12: Software development tools	92
Figure 4.13: Programming technologies	95
Figure 4.14: Database management system	96
Figure 4.15: Figure 4.15: Code version control tool	98
Figure 4.16: Application server	99
Figure 4.17: Software installation environment	100
Figure 4.18: Software development tasks	101
Figure 4.19: Skills of software developers – Agree	103
Figure 4.20: Skills of software developers – Don't know	104
Figure 4.21: Skills of software developers – Disagree	105
Figure 4.22: Other skills of software developers	108

Figure 4.23: Technologies used by interviewed the software developers	113
Figure 6.1: Software developer competency framework (SDCF)	143
Figure 6.2: Competency Cycle	155

LIST OF TABLES

Table 4.1: Response status of questionnaire	80
Table 4.2: Gender	82
Table 4.3: Age in years	83
Table 4.4: Race of software developer respondents	84
Table 4.5: Citizenship of respondents	85
Table 4.6: Work experience in years	86
Table 4.7: Time taken to become self-dependent	87
Table 4.8: Time taken to become self-dependent	87
Table 4.9: Education background of respondents	88
Table 4.10: Current level as a software developer	89
Table 4.11: Province of respondents	90
Table 4.12: Software development IDE	92
Table 4.13: Programming technologies	94
Table 4.14: Database management systems	96
Table 4.15: Code version control tool	97
Table 4.16: Application server	99
Table 4.17: Software installation environment	100
Table 4.18: Software development tasks	101
Table 4.19: Skills of software developers - Agree	102
Table 4.20: Skills of software developers – Don't know	103
Table 4.21: Skills of software developers – Disagree	104
Table 4.22: Other skills of software developers	107

APPENDICES

Appendix A: Questionnaire	169
Appendix B: Introductory letter to the industry	177
Appendix C: Letter to CPUT internship coordinators	178
Appendix D: Email to software developers	179
Appendix E: List of companies requested to participate in this research.....	180

GLOSSARY

Acronym	Explanation
AMS	American Medical System
API	Application Programming Interfaces
B2B	business-to-business
B2C	business-to-consumer
C#	C sharp (programming language)
C++	C plus plus (programming language)
CIO	Chief Information Officer
CITI	Cape Information Technology Initiative
CMM	Capability Maturity Model
COBOL	Common Business Oriented Language
CORBA	Common Object Request Broker Architecture
CPU	central processing unit
CSS	Cascading Style Sheet
DB2	Database two
DBMS	Database Management System
DCOM	Distributed Component Object Model
EJB	Enterprise Java Beans
FORTRAN	Formula Translation (programming language)
GABEK	Ganzheitliche Bewältigung Sprachlich Erfasster Komplexität
HCI	Human Computer Interface
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
ICT	Information and Communication Technology
IDE	Integrated Development Environment
jQuery	JavaScript Library
JSON	JavaScript Object Notation
MVC	Model View Controller
NIH	Not Invented Here (syndrome)
OOP	Object Oriented Programming
ORM	Object Relation Mapping
PASW	Software package used for quantitative data analysis
P-CMM	people Capability Maturity Model
PHP	Hypertext Pre-processor (programming language)
RAM	Random Access Memory
RCPSC	Royal College of Physicians and Surgeons of Canada
ROI	Return On Investment
SDCF	Software Developer Competency Framework
SDLC	Software Development Life Cycle
SEI	Engineering Institute
SOA	Service oriented architecture
SOAP	Simple Object Access Protocol
SQL	Structured Query Language
SRS	software requirements specification
TFS	Team Foundation Server
UDDI	Universal Description, Discovery and Integration
UML	Unified Modelling Language
URI	Uniform Resource Identifier
V&V	verification and validation
W3C	World Wide Web Consortium
WCF	Windows Communication Foundation
WS-BPEL	Web Services Business Process Execution Language
WSDL	Web Services Description Language
WWW	World Wide Web
XML	eXtensible Markup Language

CHAPTER ONE: RESEARCH INTRODUCTION

"If debugging is the process of removing bugs, then programming must be the process of putting them in."

(Edsger W. Dijkstra)

1.1 Introduction

Following the expansion of Information and Communication Technology (ICT), software systems are part and parcel of most of activities performed by human beings. Many activities that used to be conducted and performed manually are becoming automated by using software systems (Bosch, 2009:192; Ruxwana, Herselman & Conradie, 2010:17; Cloete, 2011:17). Typical examples are evident in e-Banking where software systems are used to facilitate banking services, in e-Health to facilitate healthcare services, in e-Commerce to facilitate online business transactions, in e-Government to facilitate government services and in e-Learning to facilitate the teaching and learning process. Software developers develop software systems for both small and large business organizations. The developed software systems however must satisfy user needs in order for business objectives to be met. While many software projects are conducted, it appears that only a few are successful (Galorath, 2009). Furthermore, the shortage of skilled software developers continues to impact the ability of the software industry to deliver (Roodt & Paterson, 2009:195-196).

With the need of using ICT to conduct business activities, software developers have important roles to play to ensure that good quality, robust, secure, safe, and user friendly software systems are developed. This research is about competencies of software developers. In section 1.2, the background to the research problem is discussed followed by statement of the research problem and research question, sub-questions and objectives in section 1.3 and section 1.4 respectively. Literature survey is introduced in section 1.5 and discussed in detail in chapter two. Research design is introduced in section 1.6 and discussed in detail in chapter three. Section 1.7 indicates the delineation of research while section 1.8 shows contribution of the research. The chapter ends with chapter summary in section 1.9, thesis structure in section 1.10 and conclusion in section 1.11.

1.2 Background to the research problem

Software can be defined as a program that accepts inputs and converts those inputs into outputs, by using a specific algorithm in order to solve a particular problem. However, in the real world software is more complex than just a program. According to Sommerville (2007:5), software is not only the computer program but also includes all associated documentation and configuration data required in order for the program to function correctly. Satzinger, Jackson and Burd (2004:36) emphasize that the program comes much later in the development process. It is important for software developers to view software as more than just a program. System documentation to describe the structure of the system and user documentation to provide information to users as how to use the program are part and parcel of software. A software developer requires outstanding competency, skills and knowledge to develop software that addresses real world problems.

Software projects can be categorized into three major groups: successful, challenged and impaired projects (Yeo, 2002:241). Similarly, Stepanek (2005) classifies software projects into three categories namely: successful, challenged and failed projects. Successful projects are completed on time, within budget limits and deliver customer needs at the required quality. Challenged projects are completed but are over budget, over the estimated project duration or provide only some of the required functionalities. Failed projects are never completed; instead they are abandoned or closed down. While a project may be completed on time, within budget and delivering quality functionalities as per the set out requirements, they may not be accepted by users, which hamper the usability and productivity of the software products (Yeo, 2002:241). The usage of software does not necessarily suggest the acceptability of the system, however, it could be due to the fact that users have no alternative hence are forced by the circumstances to accept the product (Yeo, 2002:242).

While many software projects are conducted, only a few are rated as being successful. The Standish Group (2001) reports about the studies on software projects, which indicate that only 28 percent of software projects were considered as successful, while 49 percent were challenged, and 23 percent failed. The number of challenged and failed projects continues to outweigh the number of successful projects (Agarwal & Rathod,

2006). In a study comprised of 250 software projects, Jones (2004b:5) reports that only 25 were categorized as successful; met their schedule, cost and quality objectives, 50 were delayed and 175 experienced major delays or were terminated without completion. Furthermore, Franco (2012) reports that in a survey on software development in South Africa with 186 respondents, only 48.30 percent of respondents indicated that their software development projects were successful. This implies that more than half of respondents consider their projects as a failure. Software development continues to be a challenging field where more efforts are still required in order to bring about success in software projects.

According to Nieman and Bennett (2006:285), the advent of Internet technology has contributed to the increase in the use of software systems, especially in the business world. Software is not only ubiquitous but also part and parcel of both small and large business organizations. However, poorly developed software is a major contributing factor to the increase of security threats on the Internet (Viega & McGraw, 2002:3). According to Hook (2000:14), security threats such as denial-of-service attacks, viruses, worms and Trojan horses are major concerns that businesses have to deal with regularly. Hook (2000:15) further report that many companies, including Microsoft Corporation spend millions of dollars a year for testing systems in order to discover and address software vulnerabilities that can be prone to attacks. Microsoft, for instance, has a specialized team of software engineers whose main job is to investigate attack techniques used by hackers. The team investigates hacking techniques to accumulate hacking knowledge that can be used by software developers to avoid vulnerabilities in order to prevent future system attacks (Hook, 2000).

According to Viega and McGraw (2002:11) software security is considered as a separate feature within the software development life cycle. This results into production of software with holes and vulnerable to attacks. Shreyas (2002) mentions that current software development processes incorporate security in the system once all functional requirements have been addressed. Furthermore, Shreyas (2002) mentions that the focus of software developers tends to be on functional requirements and meeting deadlines within budget. Security tends not to be a priority for software developers even in cases where security threats are easily noticeable. This has led to production of poorly written software with many security vulnerabilities, which is a great opportunity for

malicious hackers and crackers. McGraw (2002) argues that security is an emergent property and not a feature that can be pasted onto the system. Implementing security features once software has been developed requires enormous architectural and coding changes. The process of incorporating changes can be a hundred times more expensive than if security is built into the software during the software development process (Gegick & Williams, 2007:381). Many changes can introduce more security flaws resulting in the increase of software vulnerabilities. Therefore, software developers have great roles to play to ensure production of safe, secure, robust and vulnerability free software systems.

Lucia and Lepsinger (1999:5) describe competency as a tool that can be used to identify skills, knowledge, personal characteristics and behaviour associated with efficient performance of a given career. This is required to effectively execute a role in an organization in order for the business to meet its strategic goals. Mirabile (1997:21) defines competency as knowledge, skills, ability or characteristics that can be associated with high performance on a given job. Gangani, McLean and Braden (2004:1111) define competency as a description of skills, knowledge, behaviours, personal characteristics and motivations associated with success in a job. Turley and Bieman (1995), mention that job competency is a contributing factor to the success of a given job. Turley and Bieman (1995:25) further define job competency as any attribute that contributes to doing a specific job well such as knowledge, ability, interest and motivation. In this research a Software Developer Competency Framework (SDCF) is defined as a structure comprised of tasks, tools, skills, knowledge and attributes that are essential for the effective performance and success of software development activities.

With reference to Satzinger *et al.* (2004:13), many job titles are used interchangeably to describe a person dealing with production and development of software. While there are different specialities in software projects many experts share common tasks. Depending on the organization structure, job titles such as system developer, systems architect, webmaster, software engineer, systems consultant, software developer, software programmer, web developer, systems designer and many others are used interchangeably to refer to similar job titles with similar responsibilities. In this research, however, the title software developer is used to refer to any individual whose main job is to engineer and develop software systems.

A software developer requires specific skills and knowledge in order to develop software that solves problems in order to deliver intended solutions. This is a challenge to education institutions, as providers of software developers, whose main responsibility is to inculcate required skills and knowledge in order for students to become qualified and competent software developers. Likewise, an individual, who is passionate about software development as a career, should be familiar with skills and knowledge that are required in order to function effectively as a competent software developer. A person can then work on required areas of skills and knowledge in order to develop essential competencies to become a competent software developer. With reference to Figure 1.1, the primary objective of this research is to develop a competency framework for software developers.

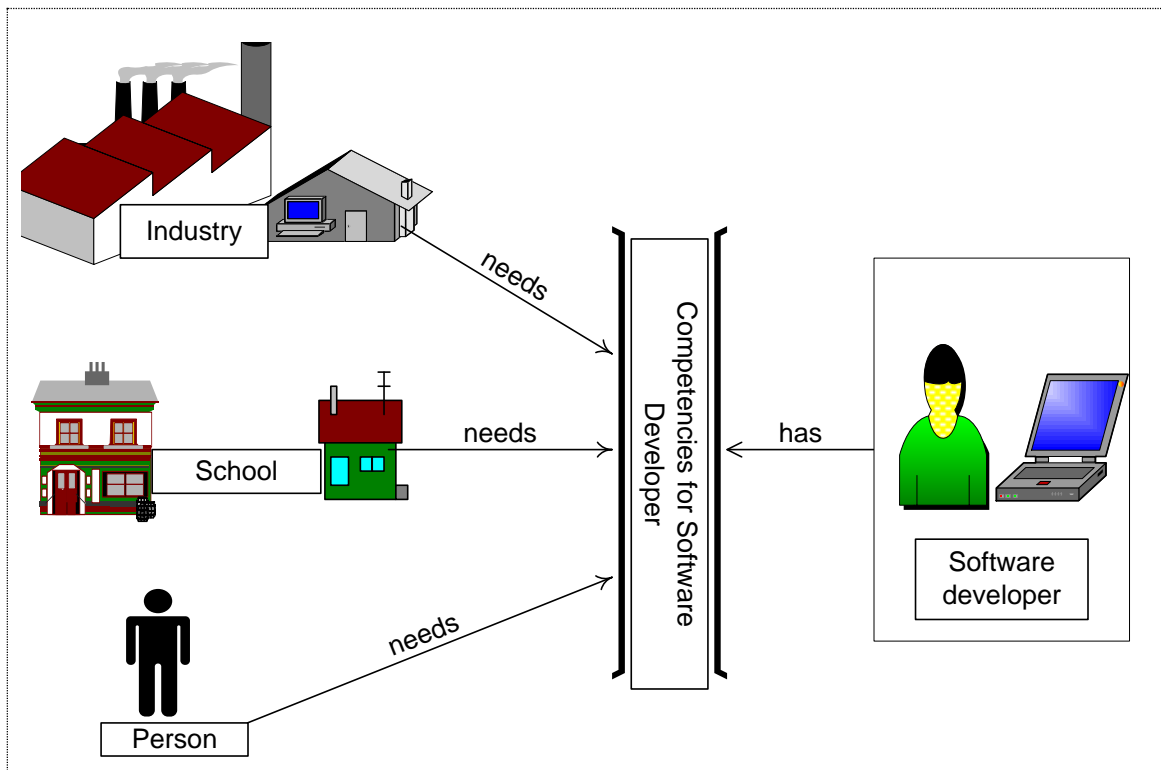


Figure 1.1: Importance of Competency Framework for Software Developers

A software competency framework is not only important to education institutions but also to the industry as the employer of software developers. As it is the goal of institutions to provide better education, it is in the best interest of software and business companies to employ competent software developers who can do their job well.

Nieman and Bennett (2006:246) mention that one of the main duties of human resource departments is to perform job analysis in order to determine substance, demands and responsibilities of a job. This is done by collecting relevant information about the job; prepare a job description, which consists of job requirements and matching the job description to job specification. Skills, knowledge and ability are the main ingredients of a job description. Hence, a competency framework for software developers can assist human resources management to determine the suitability of an employee as a competent software developer.

The Software Engineering Institute (SEI) developed several frameworks such as P-CMM (People Capability Maturity Model) and CMM (Capability Maturity Model). P-CMM is about improving ways the organization can manage its human resources. P-CMM focuses on improving the management of people since it provides a framework for motivating, recognizing, standardising and improving good practice (Sommerville, 2007:609). CMM is a framework for assessing the capabilities of contractors who develop software for other organizations (Sommerville, 2007:680). SEI was established to improve the capabilities of the USA software industry and it focuses on large projects for large organizations. Sommerville (2007:668) reports that large projects spend more time on integration, project management and communication in order to understand many parts of the software system than on developing activities. Nevertheless, small projects spend more time on designing and programming of software systems. While CMM and P-CMM are useful guides to improve production of high quality software, these models are designed for large organizations rather than for small organizations. In addition, the complete application of these models is very expensive and unnecessary to most organizations (Sommerville, 2007:609). Therefore, the practical applications of P-CMM and CMM are very limited in the South African context where most of software development is done by small organizations.

It is not uncommon for business organizations to use competency frameworks. Roles of medical experts evolve at a quick pace following continuous changes in biological, social and clinical sciences (Frank, 2005). In order to address constantly changing demands for medical professions, Frank (2005), reports about a project initiated by the Royal College of Physicians and Surgeons of Canada (RCPSC). The purpose of the project was to

identify competencies of medical professions including physicians and surgeons which resulted in the creation of a competency framework for medical experts. Frank (2005) mentions that a competency framework is being used by practicing physicians as a resource for professional development such as self-assessment and evaluation to ensure highest standards and highest quality of health care. Furthermore, Frank (2005) finds that a competency framework is being used by educators and teachers as a standard for education delivery to ensure that physicians are equipped with essential competencies for effective medical practice and health care.

With reference to the Standish Group (2001), lack of competent staff is among the recurring factors depicted in most reports about failure of software projects. According to Galorath (2009), the Standish Group report of 2009 on software projects indicates that only 32 percent of software projects were successful while 44 percent are challenged and 24 percent failed. According to Serpell and Ferrada (2007:587), developing countries suffer from inadequate number of trained workers in a large scale. Caldo (2008) and Calitz (2011) report about business organizations in South Africa struggling to find professionals with the required ICT¹ skills sets. Furthermore Roodt and Paterson (2009:195-196) report of skills shortage for computer professionals, predominantly systems analysts and programmers also known as software developers, as a problem that will continue to be experienced, in South Africa, if graduate output does not increase.

While there are many categories of staff involved in software projects, the key staff who ensure the engineering and development of the software are software developers. According to Turley and Bieman (1995:1), the attributes of software developers are the most important factors determining the success of software development. The quality of software reflects attributes, skills and knowledge of software developers involved in the engineering and development processes of the software. Colomo-Palacios, Casado-Lumbreras, Soto-Acosta, García-Peñalvo and Tovar-Caro (2013:456) agree that the quality of software products depend on knowledge and ability of software developers.

¹ ICT: Information and Communication Technology

A competency framework for software developers can help to determine and identify essential attributes and skills of competent software developers.

1.3 Statement of the research problem

As reflected in the background section it is revealed that most software projects fail and that the increase in software security problems is largely due to poorly developed software. Knowledge and information technology are key factors for survival in the competitive business world (Ahn & McLean, 2008:542). Business organizations need to utilize the best and the most information technology can offer in order to have a competitive advantage in the business world. The large number of failing software projects and the increase in software security problems are the major obstacles in the software development industry. Among others, the solution can be achieved by improving the competency of software developers so that software developed are of good quality, safe, robust and support business objectives. Therefore determining the essential competencies of software developers is vital, if an enterprise has to have suitable and competent employees as software developers and if education institutions are to produce suitable and competent candidates for software development activities. Hence the purpose of this research which is to identify and determine essential competencies of a software developer in order to develop a Software Developer Competency Framework (SDCF).

1.4 Research question, sub-question and objectives

Research Problem	The large number of failing software projects and the increase in software security problems are the major problems in the software development industry. As such, there is shortage of skilled software developers. Among others, the solution can be achieved by improving the competency of software developers so that software developed are of good quality, safe, robust and support business objectives. However, the challenge is to determine essential competencies of a software developer.	
Research Question	What are the fundamental competencies of a software developer?	
Research sub-question	Research methods	Objectives
How is software developed?	Literature study, survey and interviews	Understand which steps are involved in the software development process in order to determine tasks performed by software developers.
What technologies and tools are used to develop software?	Literature study, survey and interviews	Understand which technologies and tools are used to develop software systems.
What are essential skills required in order to develop software?	Literature study, survey and interviews	Determine skills and knowledge required by software developers

1.5 Literature survey

In this section, the work environment of a software developer is explored. This is in order to determine which software components regularly interact with software developers. As such, the Software Development Life Cycle (SDLC) is briefly examined to determine

tasks performed by software developers. In conclusion, challenges of a software developer, software reuse and software project management are briefly discussed.

1.5.1 The environment of a software developer

The need for software emanates from the need to solve a particular business problem in order to achieve specific business objectives. Satzinger *et al.* (2004:4) mention that one of the major goals of a software developer is to solve business problems. The environment of a software developer in the business world is complex. With reference to Figure 1.2, Satzinger *et al.* (2004:7) describe interacting components of information systems as inputs, output, hardware, software, data, procedures and people.

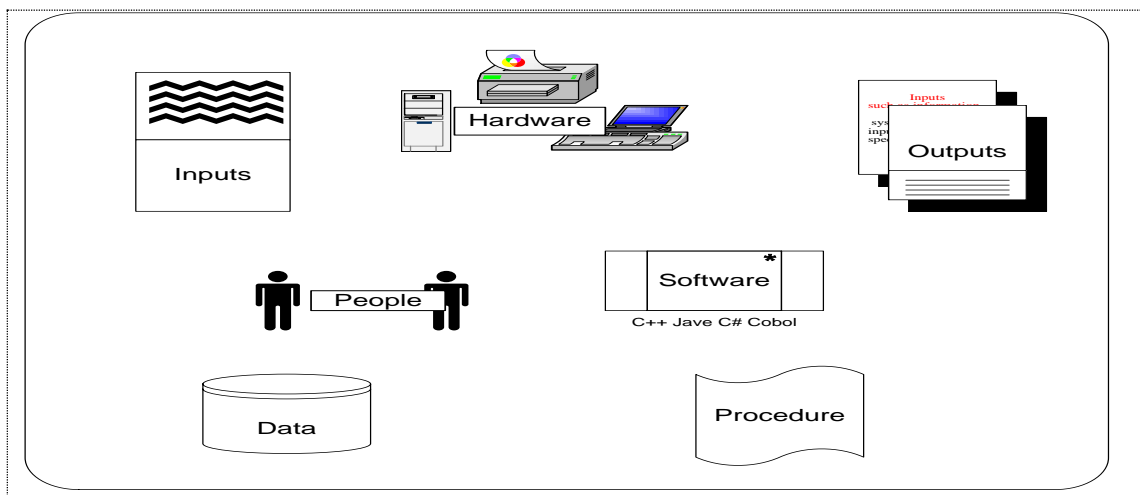


Figure 1.2: Components of information systems (adapted from: Satzinger *et al.*, 2004:7)

These components constitute the work environment of a software developer. The competency of a software developer can therefore be investigated by examining components of information systems in order to determine activities performed by a software developer.

1.5.1.1 Inputs

Software for information systems requires inputs to perform specific processing. Inputs are usually channelled to the system by actors; users or other external systems. The point of contact is required in order for the system to accept inputs. This is commonly known as an interface (Bennett, McRobb & Farmer, 2006:440). The interface can be

between users and the system, which is known as human-computer interface (HCI), or between one system and another which is called system interface. A software developer has responsibilities to ensure that the system has interface to be able to interact with actors. It is thus important to understand competencies required by a software developer in order to develop input-to-system interface.

1.5.1.2 Output

Once inputs have been processed, outputs are normally produced. Outputs could be stored, delivered to another system, displayed to users or even used as inputs for further processing in order to produce other outputs (Klein & Ralya, 1990:5-7). Depending on the purpose of the system processing, a software developer has responsibilities as far as system outputs are concerned. Hence, it becomes paramount to establish what techniques and tools are required when modelling and developing system outputs.

1.5.1.3 Hardware

According to Malik (2004:3), major hardware components are central processing unit (CPU), random access memory (RAM) and input/output devices such as keyboard, mouse, screen and printer. In simplicity, hardware could refer to any part of the system that is physical and tangible. It is important to establish skills and knowledge required for a software developer to effectively utilize hardware involved in the software development process.

1.5.1.4 Software

This section refers to software as computer programs; which is the core of software development. According to Malik (2004:6) software are programs written to perform specific tasks. Malik (2004:6) further mentions that there are two types of programs namely system programs and application programs. System programs such as the operating system control the computer while application programs such as word processors perform specific tasks. Nevertheless all programs: desktop applications, internet applications, mobile applications and operating systems, are written by software developers using programming languages. Hence, it is important to establish the

competencies required by a software developer in order to write software (computer programs).

1.5.1.5 Data

According to Rob and Coronel (2004:7), data is raw information that has to be processed in order to provide information that can be used by users. Data are inputs to computer programs and data can be received from users, files, XML (eXtensible Markup Language) documents and databases. Morrison and Morrison (2003:633) mention that databases are essential for dynamic websites such as e-commerce applications. Computer programs and data are inseparable. Hence, it is important to establish skills and knowledge required for a software developer to deal with data used in a software program.

1.5.1.6 Business procedure

Business procedures determine business policies, regulations or constraints that must be adhered to during the execution of business operations (Leymann & Altenhuber, 1994). Business procedures describe how an enterprise will achieve its business goals. User requirements, which translate into system requirements, are sources of business procedures that must be implemented by software. According to Van Vliet (2007), software must support users effectively. Violation of business procedures can be disastrous to an enterprise by changing its business goals. Leymann and Altenhuber (1994) explained that information technologies used by an enterprise must support its business procedures. It is imperative therefore to establish skills and knowledge required by software developers to translate correctly business procedures into software.

1.5.1.7 People

Olson (2004:12) mentions that poor user involvement is among the top reasons for failure of software projects. Sommerville (2007:363) recommends that users need to be involved in the software design process for the software system to achieve its full potential. Real world projects are complex and require teamwork where each team member has roles to play. Olson (2004:12) mentions that top management support is also among top reasons for failure of software projects. Due to competition for scarce

resources among projects, a software developer should be able to convince management for resources. Users, team members and management are key elements for the success of a project. It is therefore essential to establish skills required by software developers in order to effectively deal with people involved in the software project.

1.5.2 Software Development Life Cycle

According to Satzinger *et al.* (2004:36), a software development process is divided into different phases that form a software development life cycle (SDLC). There are many methodologies, approaches or processes used to execute phases of software development life cycle during a software project. These methodologies range from the primitive waterfall methodology to the advanced incremental and iterative methodologies to software development (Hughes & Cotterell, 2006:75-89). Each methodology has its advantages and disadvantages. The selection of a methodology however, depends on the nature and complexity of the software project. For instance due to the risky, uncertain and complex nature of large software projects, the iterative, prototyping and incremental methodologies are preferred to primitive methodologies such as the waterfall methodology (Hughes & Cotterell, 2006:75-89). Iterative approach is ideal for dealing with risks while prototyping is suitable for addressing uncertain situations in a software project, such as unclear user requirements.

The waterfall methodology divides software development activities into distinct tasks or phases where one phase continues after one phase has finished. For example, design of the system only begins once the user requirement phase is completely done and closed. The waterfall methodology is thus structured but not appropriate for large projects. However, it is this structured organization of waterfall that makes it ideal for learning purposes in order to understand distinct activities involved in each phase of a software project. With reference to Figure 1.3, the waterfall model is divided into five phases namely: requirement analysis, system and software design, implementation and unit testing, integration and system testing as well as operation and maintenance (Sommerville, 2007:66). While there are many methodologies, fundamental activities as indicated by the waterfall methodology, are common to all other methodologies. This research therefore utilizes the waterfall methodology as a generic model to determine

activities performed by software developers in each phase of a software development life cycle.

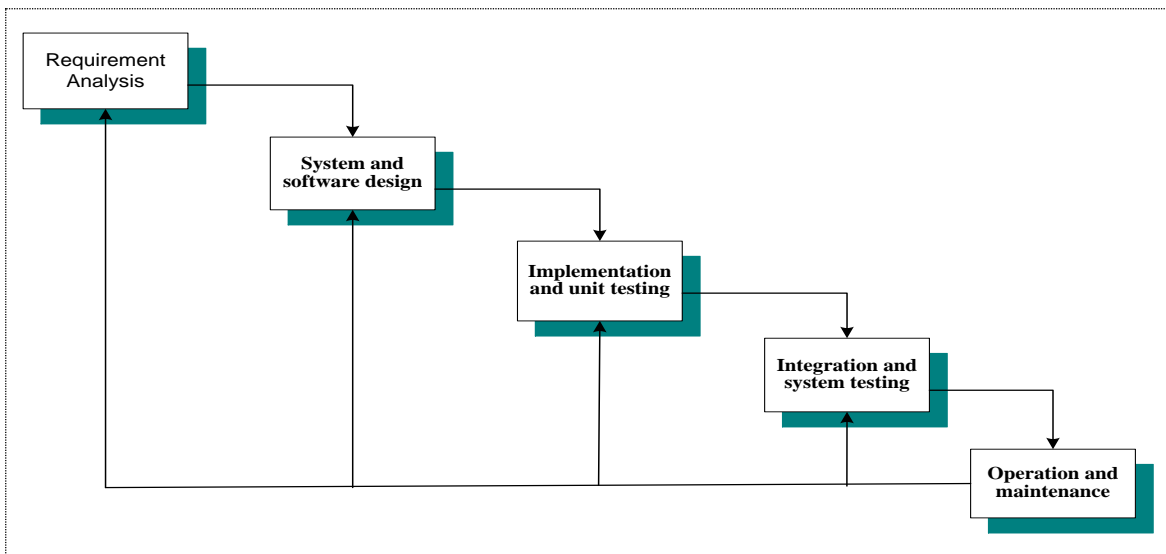


Figure 1.3: The waterfall model (adapted from: Sommerville, 2007:66)

1.5.2.1 Phase 1: Requirement analysis

The goal of an information system is to satisfy specific user needs (Satzinger *et al.*, 2004:6). A software developer should develop software that satisfies user requirements. Hence the objective of the requirement analysis phase which is to understand and establish user requirements. According to Sommerville (2007:118), the process of finding out, analysing and documenting user needs is called requirement engineering. Some of techniques that a software developer might use to determine user requirements include gathering information by interviewing users, defining system requirements and building prototypes in order to discover user needs. During this phase, a software developer has to work closely with stakeholders: people who will be using or affected by the system to be developed. A software developer must understand the overall objectives of the business and what users want to achieve in their jobs (Bennett *et al.*, 2006:129).

According to Sommerville (2007:119-127) information gathered during the requirement phase falls into three categories namely: functional requirements, non-functional requirements and domain requirements. Functional requirements describe what the system is expected to do in order for users to do their jobs. Non-functional requirements

refer to how well the system performs what is supposed to do. This includes system characteristics such as system performance, reliability, usability, availability, security and others. These are emergent properties of a system since they can be determined once the system is completed. Domain requirements are system requirements pertaining to the application domain of the system to be developed. Domain requirements are business specific, for instance a standard user interface for all applications in a particular domain. Poor requirements are among the major contributors of failure in software projects (Hofmann & Lehner, 2001). Users are very important during requirements analysis hence software developers must work intensively with users in order to understand what users want the system to perform.

1.5.2.2 Phase 2: System and software design

After determining user needs, a software developer must establish how those user needs will be achieved by the system to be developed. Instead of immediately starting developing the system, a software developer must design a system. Bennett *et al.* (2006:372) describes design as a process of developing a system without building it. According to Bennett *et al.* (2006:371), during design phase, a software developer is concerned with how the system will address user needs. Bennett *et al.* (2006:376), further describes two levels of design namely: system design and detailed design. System design is about the construction or architecture of the overall system such as the human-computer interface. On the other hand, detailed design is concerned with designing individual components of the system. Although not on a large scale as in analysis phase, users must be involved and informed of the developments in the design phase. For instance, user's feedback regarding the design of the human-computer interface is vital to the usability and acceptability of the system by users (Sommerville, 2007:378-385). Some of the major activities that take place during design phase include design of the system architecture, design of user interfaces, design and integration of database and prototype for design details (Satzinger *et al.*, 2004:39).

1.5.2.3 Phase 3: Implementation and unit testing

In this phase, the actual components of the system are built. Each component is tested to verify that it meets its specification. According to Sommerville (2007:547), unit testing is a process of testing individual components in the system. This process is crucial in

order to expose defects, flaws and faults within built system components. Satzinger *et al.* (2004:40) mentioned that construction of software components, unit testing and verification are some of major activities performed during this phase.

1.5.2.4 Phase 4: Integration and system testing

During this phase, individual developed components of the system are integrated in order to work together (Sommerville, 2007:540). In addition, the software system is tested as a whole to ensure that software requirements have been met. System testing can lead to acceptance testing where customers are involved to ensure that user requirements are met (Sommerville, 2007:541).

1.5.2.5 Phase 5: Operation and maintenance

According to Sommerville (2007:67), operation and maintenance phase is the longest phase. It is during this stage when a system is installed and put into practical use. Some of activities during this phase are correcting errors, which were not discovered during earlier phases, improving the implementation of system units, and enhancements to system services as new requirements materialize. Satzinger *et al.* (2004:41) referred to this phase as a support phase. Satzinger *et al.* (2004:41) further mentioned that system maintenance, system enhancement and supporting system users are critical elements during support phase.

1.5.3 Challenges of a software developer

Hughes and Cotterell (2006:4) report that software is intangible, invisible, and complex. Project management for software project is like to make the invisible to be visible (Hughes & Cotterell, 2006:4). Software developers have to conform to user requirements, which are not necessarily consistent. Hence, software development is a challenging undertaking. It is therefore important to identify common challenges facing software developers in order to establish essential skills required to address, manage and control software development challenges.

1.5.4 Software developer and software reuse

Software reuse is the process of creating software systems from existing software rather than building software systems from scratch (Krueger, 1992:131). Bennett *et al.* (2006:223-225) cite that software developers do not utilize the benefits associated with software reuse. Bennett *et al.* (2006:224), is of the opinion that some of the reasons that software reuse has not been successful as promised is due to the NIH (Not Invented Here) syndrome. With such altitude, software developers tend not to trust other people's codes and the fact that software reuse is difficult to manage. Software developers therefore prefer to re-invent the wheel by developing software from scratch. It is thus imperative to establish competencies essential to support and promote software reuse in order for software developers to speed up software development without compromising software quality.

1.5.5 Software project management

Coordination of activities during different phases of software development life cycle can be challenging. According to Hughes and Cotterell (2006:4), software projects are complex in nature. It is the nature of the complexity that makes coordination and management of software projects cumbersome. However, there are tools and techniques available to facilitate the management process of software projects in order to achieve the envisaged project objectives. Olson (2004:5-6) reports that time, budget and quality are the main competing project dimensions. Hence, these project dimensions should be monitored closely during the software development process.

According to Olson (2004:2), a project is an endeavour or any new initiative performed in order to achieve specific goals within budget and within a given period. Hence, software development process is a project. While a software developer is responsible for executing different tasks during the development process, the challenge can be the coordination and management of tasks without interrupting the project objectives. It is thus critical for a software developer to be aware of tools and techniques available to ensure smooth execution of a software project. It is important to establish critical sections of software project that can be managed by a software developer in order to make project management easier. The aim is to determine skills required by software

developers to perform software development activities without affecting business objectives as well as project objectives such as project deadlines and project budgets.

1.6 Research design

According to Welman, Kruger and Mitchell (2005: 6-9) research methodologies can be categorized into two paradigms; namely qualitative and quantitative research. Qualitative research involves research methods where qualitative data such as descriptions of social life as observed by a researcher, unstructured interviews and data obtained from written sources are employed while quantitative research deals with quantitative data; which are numerical and hence measured and expressed in form of numbers such as a survey (Jones, 2004a). Welman *et al.* (2005:8) mention that the purpose of quantitative research is to evaluate objective data consisting of numbers whilst qualitative research deals with subjective data that are produced by the minds of respondents or interviewees, which is represented in language instead of numbers.

According to Welman *et al.* (2005:166), unstructured interviews offer a greater wealth of information than other methods of collecting data. Welman *et al.* (2005:166) also mention that unstructured interviews are usually used in explorative research because of their qualitative nature. Furthermore, Welman *et al.* (2005:166) report that unstructured interviews methodology is ideal for a research whose objective is to understand the experiences of individuals in their life-world and identify important variables in a particular research area.

Krauss (2005:758) reports that the term epistemology originates from a Greek word “episteme” which mean knowledge and hence defined epistemology as the philosophy of knowledge. Krauss (2005:764) further reports that qualitative epistemology involves face-to-face interviews in order to tap into the mind of subjects to have a depth understanding of a phenomenon. The best way to understand a phenomenon is to immerse in that phenomenon, for example, by moving into the culture or organization being researched and experience to be part of the phenomenon (Krauss, 2005:760). According to Bowen (2005:211) the most popular methods for qualitative epistemology are interviews, observation and document studies. Bowen (2005:209) and Mumford

(2006:384) state that the deep and detailed understanding of a phenomenon and lived experiences is best achieved via qualitative epistemological research.

Turley and Bieman (1995:3) during phase one of their research on competencies of software engineers; they used interviews to identify essential competencies of software engineers in a software development company in United States of America (USA). In phase two, Turley and Bieman (1995:11) used survey questionnaires to cover larger samples in order to validate their research. Ahn and McClean (2008:545) used interviews to identify competencies required for port and logistics personnel in Busan, South Korea. Thereafter, Ahn and McClean (2008:547) had to use survey questionnaires to cover large samples in order to validate their research on larger samples. This implies that interviews and survey are often used together to complement competency studies.

According to Le Deist and Winterton (2005:31), competency framework is a result of observing successful and effective job performers and determining what makes these individuals to differ from less successful performers. Frank (2005) reports about intensive steps involved to identify competencies of physicians and medical doctors. Frank (2005) mentions that methodologies applied to identify and determine competencies of physicians and doctors included consultation among many medical specialties such as physicians and surgeons as well as among doctors and patients. The research was however complemented by surveys to test the validity of the competency framework.

Based on the above mentioned, it is clear that social science studies tend to use both qualitative and quantitative methodologies together; the process known as triangulation (De Vos, Strydom, Fouche & Delport, 2002:341-342). Yin (2003) defines triangulation as a process where both qualitative and quantitative methods are used to facilitate the research process for a given research study. Olsen (2004) defines triangulation as the mixing of data or methods in order to allow diverse viewpoints or standpoint to cast light upon a given topic. Olsen (2004) further sub-divides triangulation into two categories; namely data triangulation (mixing data types) and methodological triangulation (mixing methodologies). The effectiveness of triangulation is based on the premise that weaknesses in one method are compensated by strengths of another method (Jick,

1979:604). Jick (1979:602) explains that qualitative and quantitative methods are complementary and not rival methods. Furthermore, Olsen (2004) reports that mixing methodologies such as the use of survey and interviews is a more profound form of triangulation. As such, this research uses survey questionnaires (Appendix A) and interviews. Both survey questionnaires and interviews aim at obtaining information pertaining to tasks performed by software developers, tools used and skills required to become a competent software developer. Therefore this study is accomplished via both quantitative and qualitative epistemological methodologies.

1.7 Delineation of the research

This research focuses on software development activities performed by software companies and business organizations within the Western Cape Province. While there are many categories of software developers such as developers for scientific applications, robotics applications, embedded systems and many others, this research deals specifically with software developers involved with the development of business applications to solve common business problems. However, the research can be applicable to any software development process, since according to Sommerville (2007:64), all software processes have common fundamental activities.

1.8 Contribution of the research

The contribution of this research emanates from the fact that very little has been done regarding research on competencies of software developers particularly in South Africa. The researcher could not find any research that deals with a competency framework specific to software developers in South Africa. Turley and Bieman (1995:1) researched competencies for software engineers for a software development company in the United States of America (USA). Another study was carried out in USA on master's students of computer science who were assigned software development tasks (Rivera-Ibarra, Rodriquez-Jacobo, Fernandez-zepeda & Serrano-Vargas, 2010:35). Nevertheless, no research has been done on software developer competencies to address business needs specific to the South African context. Moreover the above mentioned studies are not specific to software developers as they include other professionals involved in the software engineering process of a given software system. Hence, the main objective of

this research is to develop a competency framework specific for software developers from the South African business perspective.

A competency framework as an analysis tool can be used to identify performance gaps within employees and suggest measures to improve employees' performance. Competency frameworks can be used to develop training plans for an enterprise in order to improve performance of less productive employees. Serpell and Ferrada (2007:585) report on the effectiveness of a competency framework for training, developing and certifying supervisors in the building and construction sector in Chile, Southern America. According to Serpell and Ferrada (2007:599), competency frameworks can be effective by helping human resources achieve skills and knowledge required to perform specific functions. Competency framework can be useful in developing countries where the number of inadequately trained workers is large (Serpell & Ferrada, 2007:587). Competency frameworks can be used by human resource management for selection of suitable candidates and staffing of competent employees. According to Gangani *et al.* (2004:1112-1113) the American Medical System (AMS), a company developing, manufacturing and marketing medical technologies, used a competency-based framework to improve the performance of its human capital. Ahn and McLean (2008:544) report that a competency framework is a tool that can help to manage and improve personnel performances. The framework can be used by business organizations to determine competency and suitability of an employee to work as a software developer. According to Brooks (1987), using great designers is a positive step to improve software development productivity and Boehm (1983) recommends that better and fewer people are suitable for software projects.

According to Frank (2005) roles of medical experts evolve at a quick pace following continuous changes in biological, social and clinical sciences. In order to address constantly changing demands for medical professions, Frank (2005), reports on a project initiated by the Royal College of Physicians and Surgeons of Canada (RCPSC). The purpose of the project was to identify competencies of medical professions including physicians and surgeons which resulted into a creation of a competency framework for medical experts. Frank (2005) reports that the competency framework is used by practising physicians as a resource for professional development and for self-assessment and evaluation so as to ensure highest standard and quality of health care.

Frank (2005) also reports that the competency framework is used by educators and teachers as a standard for education delivery to ensure that physicians and medical doctors are equipped with essential competencies for effective medical practice and health care. Therefore, a software developer competency framework can be used by education institutions to prepare a curriculum relevant to the industry requirements. Candidates can use the framework to determine their competencies and identify areas that require improvement to become competent with software development activities. Furthermore, competency framework can be used to develop software tools that can be used to increase productivity of both novice and experienced software developers. In addition, the contribution of this research includes the expansion of the body of knowledge regarding competencies of software developers.

1.9 Chapter summary

This chapter discussed the background to research, the research problem, research question and sub-questions and introduced literature review pertaining to software development and competency studies. The objective of the research and research design are explained. The chapter ended by indicating delineation to the research and contribution of the research.

1.10 Thesis structure

Figure 1.4 in the following section, illustrates the structure of this thesis which is divided into six logical chapters as follows:

i. Chapter One: Research Introduction

Introduces the research, discusses problem statement and research objectives.

ii. Chapter Two: Literature Review

Discusses literature review on software development and competency studies.

iii. Chapter Three: Research Methodology

Discusses research methodology followed during this study.

iv. Chapter Four: Data analysis and Presentation

Presents results of data analysis performed during this study.

v. Chapter Five: Research Discussion

Discusses results of data analysis and research findings.

vi. Chapter Six : Recommendations and Conclusion

Provides recommendations and conclusion reached during the research.

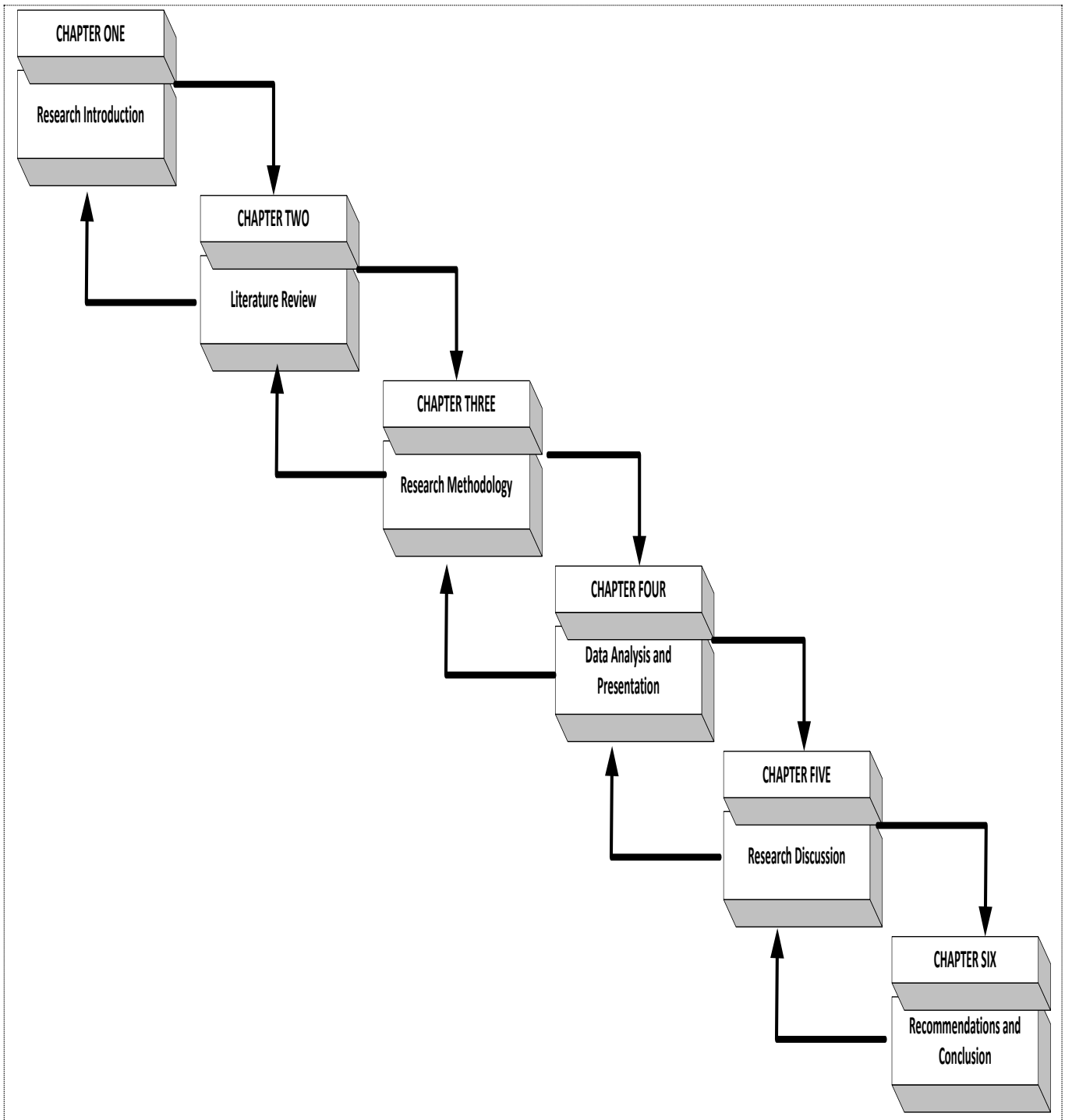


Figure 1.4: Thesis structure

1.11 Conclusion

Knowledge about Software Developer Competency Framework (SDCF) is essential if education institutions have to train learners to become future competent software developers. Similarly, it is paramount for the industry to employ software developers who can do their job well in order to address business challenges. Moreover people who would like to have a carrier as a software developer should be able to understand the level and kind of competencies required to become a software developer. Following the astonishing rate of failure in software projects and shortage of skilled software developers in South Africa, it is critical that essential competencies of software developers are identified and inculcated among software developers. Hence the objective of this research which is to develop a Software Developer Competency Framework (SDCF).

The following chapter two discusses literature review.

CHAPTER TWO: LITERATURE REVIEW

"Life can only be understood backwards; but it must be lived forwards"

(Soren Kierkegaard)

2.1 Introduction

In this chapter, literature related to competencies in software projects is discussed.

The chapter explores software projects, recent studies on competencies and the software development process of software systems. This is an attempt to understand skills applicable to software developers in the software development field in order to establish essential competencies required by software developers to perform their tasks well. According to Bennett *et al.* (2006:141), successes of system development projects, among others depend on skills of analysts, designers and programmers; referred to as software developers in this research. Turley and Bieman (1995:1) report that competencies of software developers are important factors determining success of software development projects. Furthermore, Colomo-Palacios *et al.* (2013:456) agree that the quality of software products depend on knowledge and ability of software developers.

The chapter commences with an investigation on general situation of software projects in section 2.2. In section 2.3 recent studies on competencies is explored followed by a discussion on the software development process in section 2.4. Finally, the chapter ends with chapter summary and conclusion in section 2.5 and section 2.6 respectively.

2.2 General situation of software projects

Software projects can be categorized into three major groups: successful, challenged and impaired projects (Yeo, 2002:241). Similarly, Stepanek (2005) classifies software projects into three categories namely: successful, challenged and failed projects. Successful projects are completed in time, within budget limits and deliver customer needs at the required quality. Challenged projects are completed but are over budget, over the estimated project duration or provide only some of the required functionalities. Failed projects are never completed instead they are abandoned or closed down. While a project may be completed on time, within budget and delivering

quality functionalities as per the set out requirements, they may not be accepted by users, which hamper the usability and productivity of the software products (Yeo, 2002:241). The usage of software does not necessarily suggest the acceptability of the system, however, it could be due to the fact that users have no alternative hence are forced by the circumstances to accept the product (Yeo, 2002:242).

The Standish Group (2001) reports that 23 percent of software projects failed, 49 percent were challenged and only 28 percent were considered as successful. Lack of competent staff is among the recurring factors depicted in most of reports about failure of software projects (Standish Group, 2001). According to Galorath (2009), the Standish Group report of 2009 on software projects indicates that only 32 percent of software projects were successful while 44 percent are challenged and 24 percent failed. The number of challenged and failed projects continues to outweigh the number of successful projects (Agarwal & Rathod, 2006).

According to Serpell and Ferrada (2007:587), developing countries suffers from inadequate number of trained workers in a large scale. Calldo (2008) and Calitz (2011) reports about business organizations in South Africa struggling to find professionals with the required ICT skills sets. Furthermore, Roodt and Paterson (2009:195-196) report of skills shortage for computer professionals, predominantly systems analysts and programmers, as a problem that will continue to be experienced in South Africa, if graduate output does not increase.

In order to determine roles that a software developer can play to ensure success in software projects, this section explores about software project characteristics, complexity of software projects and project management of software projects. Since software developers form a critical component of software projects, it is important to understand competencies required by software developers to effectively and efficiently participate and execute software projects.

2.2.2 Software project characteristics

According to Olson (2004), software projects are identified by large degree of uncertain and risk. Uncertainty and risky nature of software projects can be attributed by the fact that during a project life cycle, stakeholders can hardly guarantee what will happen as the project progress. The Standish Group (2001) reports about project size, project duration, team size and project cost as complex elements of software

projects. Most of software projects in the business world are typically large and are comprised of more than 10,000 function point in size, which is equivalent to 1,250,000 lines of codes in the C programming language (Jones, 2004b).

Among other roles, project management is essential for the planning, leading, organizing, controlling, monitoring and coordination of project activities (Standish Group, 2001). In order to provide functionalities required by users, a software project success is determined and characterized by time, budget and quality (Olson, 2004). Time refers to the duration required to complete a project while budget implies cost estimates for running project activities until completion of the project. Quality is about system functionalities and it measures the extent by which user objectives are met by system functionalities.

Software projects are comprised of a combination of computer hardware, communication technology and software in order to handle business processes (Yeo, 2002). Software projects thus demand resources in order to function; resources that tend to be scarce depending on project size. Software projects are said to be temporary in nature, only exist when activities have to be executed and expires once activities are completed (Olson, 2004). This involves formation of project team every time when a project is to be undertaken. A set of temporary activities is the primary characteristic of software projects. Because of this temporary nature, project members normally do not know each other very well. In addition, project teams tend to be comprised of different people, with different skills and interests (Olson, 2004).

According to Hughes and Cotterell (2006), products of software projects have peculiar characteristics that differ from products of other traditional projects. While working on bridge construction project, eventually the constructed bridge will be noticed and the progress is visible and tangible. On the contrary, developing a software system is not immediate. Hughes and Cotterell (2006) thus mention that software project management is like making the invisible to be visible. Hughes and Cotterell (2006) further report that software projects deal with developing software products that conform to requirement of human clients while other traditional projects conform to physical laws that are consistent. As a result software developers are presented with challenges to conform to user needs which due to human errors, ignorance or miscommunication, may be inconsistent, illogical and difficult to implement (Hughes & Cotterell, 2006).

2.2.3 Complexity of software projects

Hughes and Cotterell (2006) posit that software products are more complex than other engineered artefacts. As far as the number of people is concerned, software projects may require many people with diverse skills. The greater the number of people involved, the more demanding and complex is the management and coordination of project activities (Yeo, 2002). In addition, uncertainty is another complex nature of software projects. Uncertainty makes it difficult to determine time required to complete a particular first time undertaking of a software project. As such, most of software projects take longer than expected.

The field of software development lacks reliable models to estimate project costs and time with precision (Agarwal & Rathod, 2006). As such, cost estimates tend to be complex depending on the size and nature of the software project.

The manifestation of software complexity also appears in determining the success of a particular project. While a project may be complete in time, within budget and providing quality and functionalities as per requirements, the same project may not be accepted by users which would hamper the usability and productivity of the software product produced (Yeo, 2002). As already mentioned, the heavy usage of software produced does not necessarily suggest the acceptability of the system; however it could be due to the fact that users have no alternative and hence have to accept the product (Yeo, 2002).

The final product of a software project is a software system that will be used for running business processes. However, software is intangible as opposite to tangible products such as buildings, roads and bridges. Software projects deals with the production of soft and intangible products, which are more complex and more difficult to measure than tangible products (Stepanek, 2005). Stepanek (2005) mentioned about software complexity, software abstraction, rapid technology changes and incomplete user requirements as sources of difficulties associated with software projects. As far as software complexity is concerned, Stepanek (2005) reported that even minor software program tends to accumulate frightening complexity. A small program can require tens of thousands of lines of codes in order to function. The increases in the number of lines of codes and the increase in the interactions

between the lines of codes, increases the complexity associated with software projects.

Other artefact projects deal with development of tangible structures such as construction of a bridge or a building while software projects are dealing with development of intangible products such as information systems. Intangible products are difficult to measure with precision as such they depend on estimations; costs and benefits of software projects can only be estimated (Olson, 2004). The abstract nature of software demands the use of visualization and estimation in order to establish software project measurements such as project costs, project duration, project quality and software functionalities (Stepanek, 2005). While it could be manageable to visualize and estimate behaviours of a small software project in order to produce lines of code required, it is more difficult for larger software projects that would require tens of thousands or tens of millions of lines of code in order to function.

Software projects are usually commissioned to address the needs of customers. While customers can be experts in their own business roles, they tend not to have technical skills as software developers. Software developers have technical skills to turn the abstraction and complexity of user needs into software programs. The need for user requirements to be understood by developers is important, however, this can be challenging and it requires users and developers to work together to refine the requirements of the software to be produced, the process that could be repeated several times before the software product is developed (Stepanek, 2005). According to the Standish Group (2001), problems relating to user requirements and specifications were the top factor for challenged software projects; user requirements problems were reported to be the most significant factors for 42 percent of software projects.

According to Stepanek (2005), computers double in speed about every two years, which opens more opportunities to software developers. Following rapid technological changes, software changes quickly and the impact of those changes cannot be predicated with precision. For example, technological changes could result into making the software outdated and no longer catering for initial user needs. Rapid change in technology can limit the usefulness of previous accumulated experiences, which demands that software developers continue to update their software

development knowledge. Stepanek (2005) further reports that following rapid technology changes, every significant new piece of software is written by software developers who are novices to the task, because whatever a software developer was working in several years ago is unlikely to be of any direct use today; for instance 20 years of experience on software development using COBOL², has little if not nothing to offer to a new software project to be developed using object oriented programming languages such as Java or C-sharp (C#). According to Olson (2004), there are many excellent applications for computer technology to support business; however, the problem is the highly dynamic nature of technology. Olson (2004) reports about information systems that took years to implement which resulted into installation of a new system after it is out-dated by newer technology. The changing nature of technology makes software projects challenging. Software products are prone and subject to high degree of change. For instance, whenever a software system interfaces with any physical or organization system, the software system will have to be changed in order to accommodate other components (Hughes & Cotterell, 2006).

The uncertainty and risky nature of software projects coupled with the complexity related to estimation of time, cost and quality objectives, makes the development of software systems difficult. Nonetheless, Olson (2004) reports that client involvement, top management support and clear project objectives are the most consistent factor among successful projects. Therefore, software developers, while designing and developing software systems should put into consideration factors that determine successful software projects.

2.2.4 Project management of software projects

Considering complexity nature of software projects, software projects demand for professional level of management. The role of project management runs throughout the entire project life cycle (Jones, 2004b). According to Jones (2004b), in a study on comparison of large software projects that successfully achieved cost and schedule estimates against those that ran late, were over budget, or were cancelled without completion, the role of project management appears to be critical. In his study, Jones (2004b) reports that out of 250 software projects analysed, 175 project experienced major delays and overruns or were terminated without completion, 50 projects had delays or overruns below 35 percent and only 25 were considered

² COBOL: programming language previously widely used for developing business applications

successful in that they achieved, their schedule, cost and quality objectives. The successful projects were attributed to project management skills and experience. Jones (2004b) further mentions about the six essential project management roles as project planning, cost estimation, project measurement, milestone tracking, change control and quality control. Successful software projects do planning very well, on the contrary, challenged and failed projects are reported to have poor planning.

Many problems can be expected during the execution of software projects. Hughes and Cotterell (2006:152) report on Boehm's list of top risk items in software projects. Among others, the top risk items includes personnel shortfalls, developing the wrong software functions, developing the wrong user-interface, late changes to requirements and development technically too difficult. Olson (2004) mentions that actions should be taken throughout the project in order to monitor and ensure that project risks do not get out of control. Risk management therefore is an important area that software developers should be aware of and regularly observe. Standish Group (2001) reports that software projects would be less prone to failure and more likely to succeed with the help of competent and experienced staff. Since software developers form a critical portion of software project staff, software developers have essential roles to play for software projects to become successful.

2.3 Recent studies on competencies

As indicated in chapter one, Lucia and Lepsinger (1999:5) describe competency as a tool that can be used to identify skills, knowledge, personal characteristics and behaviour. This is required to effectively and efficiently execute a particular role in an organization for a business organization to meet its strategic goals. Mirabile (1997: 21) defines competency as knowledge, skills, ability or characteristics that can be associated with high performance on a given job. Gangani *et al.* (2004:1111) defines competency as a description of skills, knowledge, behaviours, personal characteristics and motivations associated with success in a job. Turley and Bieman (1995), mention that job competency is a contributing factor to the success of a given job. Turley and Bieman (1995:25) further define job competency as any attribute that contributes to doing a specific job well such as knowledge, ability, interest and motivation. A software developer competency framework can therefore be viewed as a structure or a tool comprised of skills, knowledge and attributes that are essential for effective, efficient and successful performance in executing software development activities.

However the field of software engineering continues to be chaotic where most of software projects are considered challenged, failed and short of skilled workers (Standish, 2001; Jones, 2004b:5; Olson, 2004:8; Agarwal & Rathod, 2006; Serpell & Ferrada, 2007:587; Galorath, 2009). Furthermore software security threats due to poorly developed software systems, remain a major concern among the online communities; business organizations and Internet users (Viega & McGraw, 2002:2; Hook, 2000:14).

While many studies on competencies have been conducted, very little has been done regarding research on competencies of software developers. Turley and Bieman (1995:1) conducted a research on competencies for software engineers for a software development company in the United States of America (USA). Another study was carried out in USA on master's students of computer science who were assigned software development tasks (Rivera-Ibarra *et al.*, 2010:35). Nevertheless, no research has been done on software developer competencies to address business needs specific to companies in South Africa.

As briefly indicated in chapter one, application of competency frameworks to improve the functioning of business organizations is not uncommon. Roles of medical experts evolve at a quick pace following continuous changes in biological, social and clinical sciences (Frank, 2005). In order to address constantly changing demands for medical professions, Frank (2005), reports on a project initiated by the Royal College of Physicians and Surgeons of Canada (RCPSC). The purpose of the project was to identify competencies of medical practitioners including physicians and surgeons that resulted into a creation of a competency framework for medical experts. Frank (2005) mentions that a competency framework is being used by practicing physicians as a resource for professional development such as self-assessment and evaluation to ensure highest standards and highest quality of health care services. Furthermore, Frank (2005) reports that a competency framework is being used by educators and teachers as a standard for education delivery to ensure that physicians are equipped with essential competencies for effective medical practice and health care services. Therefore a software developer competency framework can be used as a resource for education, professional development, assessment and evaluation among software developers

In terms of objectives, the main goal of competency studies centre around improvement of job performance among employees. A competency framework as an

analysis tool can be used to identify performance gaps within employees and suggest measures to improve employees' performance. A competency framework can be used to develop training plans for an enterprise in order to improve performance of less productive employees. Serpell and Ferrada (2007:585) report on the effectiveness of a competency framework for training, developing and certifying supervisors in the building and construction sector in Chile, Southern America. According to Serpell and Ferrada (2007:599), a competency framework can be effective by helping human resources achieve skills and knowledge required to perform specific functions well.

A competency framework can be useful in developing countries where the number of inadequately trained workers is large (Serpell & Ferrada, 2007:587). A competency framework can be used by human resource management for selection of suitable candidates and staffing of competent employees. According to Gangani *et al.* (2004:1112-1113) the American Medical System (AMS), a company developing, manufacturing and marketing medical technologies, used a competency-based framework to improve the performance of its human capital. Ahn and McLean (2008:544) report that a competency framework is a tool that can help to manage and improve personnel performances. The framework can be used by business organizations to determine competency and suitability of an employee to work as a software developer. Boehm (1983) recommends that better and fewer people are suitable for success in software projects. Brooks (1987) agree that using great designers is a positive step to improve software development productivity.

It is therefore correct to mention that despite the differences in areas under research, objectives of competency studies are geared towards improving job performance of workers in a given field. This research hence targets the job performance of software developers where the main objective is to develop a competency framework for software developers. As indicated in chapter one, Software Developer Competency Framework (SDCF) can be used by education institutions to prepare curriculum relevant to the industry requirements and addresses business needs. Most importantly, the industry can use the SDCF to prepare software developers who can address business needs. Candidates can use SDCF to determine their competencies and identify areas that require improvement to become competent with software development activities. Lastly but not the least, SDCF can be used to develop

software tools that can be used to increase productivity of both novice and experienced software developers.

Furthermore, the contribution of this research includes the expansion of body of knowledge regarding competencies in the software engineering field.

Software developers are engineers who design and manufacture software systems (Sommerville, 2007:7). Software developers have essential and critical roles to play during the software development process in all phases of software development life cycle. Understanding competencies required by software developers can play important roles in resolving challenges facing the software engineering field as far as success of software projects are concerned. Knowledge of competencies of software developers have the potential to change knowledge of software development in order to improve quality of software systems developed.

Software engineering principles are key elements underpinning this research. Solid knowledge of principles of software development is paramount if one has to understand competencies related to software development. In order to understand essential competencies required by software developers, the following section discusses the software development process.

2.4 Software development process

There are several software development processes such as waterfall methodology, incremental and iterative processes, prototyping and the spiral model (Hughes & Cotterell, 2006:74 – 89; Dorfman, 1999). Incremental and iterative approaches are often regarded as agile methodologies. Agile methodologies are preferred and largely implemented in big software projects. The reasoning behind agile approaches is based on the premise of discovering risks, uncertainties, understand the complexity of user requirements and address dynamic changes in user requirements. In agile methodologies, software systems are developed in continuous iterative manner where each iteration results into the release of functional software components deployed for clients to use immediately. This is contrary to the waterfall methodology which is typically used in small projects where user requirements are clearly defined and understood. In this case, a project is divided into distinct phases where one phase is completely done and closed before proceeding with the next phase.

The use and choice of the development process depends on the nature of the software project to be undertaken. It is however recognized that all software development processes have lots in common. In order to understand tasks performed by software developers, it is important to examine phases of the software development process or software development life cycle (SDLC). With reference to Sommerville (2007:65), waterfall methodology is structured into distinct phases that give a good structure or framework for theoretical understanding of tasks encompassing software development processes.

According to Satzinger *et al.* (2004) and Sommerville (2007:66) phases of a software development process can be categorized as, requirement analysis, system and software design, implementation and unit testing, integration and system testing as well as operation and maintenance. The following section examined each phase to identify tasks performed by software developers, techniques and tools used as well as skills required to perform identified software development tasks. This is an attempt to gather answers regarding competencies required by software developers to effectively and efficiently develop robust, safe, secure and good quality software systems.

2.4.1 Requirements analysis

There are many versions of definitions regarding requirements. Nevertheless all definitions of requirements are geared towards understanding of needs, demands or conditions that have to be satisfied in order for a given process to proceed and bring about expected changes in a system or society. Jonasson (2008) define a requirement as follows:

- i. A condition or capability needed by a stakeholder to solve a problem or achieve an objective.*
- ii. A condition or capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification, or other formally imposed document.*
- iii. A documented representation of a condition or capability as in (I) or (II).*

According to Kujala *et al.* (2001), user requirements refer to functions, constraints or other properties that must be met in order to satisfy user needs. The process of identifying, determining and documenting user needs is called requirement engineering or requirement analysis. Sommerville (2007:143) posits that the goal of

requirement engineering is to create and maintain a system requirements document. Requirement engineering is the process of establishing services required by users from a system and the constraints under which the system is developed and operates (Sommerville, 2007:143-167; Sommerville & Sawyer, 1997). The essence of requirement engineering is to determine and document what users need the system to achieve.

Poor requirements are the biggest cause of failure in software projects (Hofmann & Lehner, 2001). Users are very important during requirements engineering process hence software developers must work intensively with users in order to understand what users want the system to perform. According to Rouibah and Al-Rafee (2009) costs for correcting user requirements is lower than costs incurred to correct errors during later stages of software development. Rouibah and Al-Rafee (2009) suggest that it is important to pay more attention to user requirements analysis in order to develop successful information systems. Requirement engineering is the most critical phase of system development (Rouibah & Al-Rafee, 2009).

Once a business opportunity has been identified, an organization embarks on determining business requirements in order to invest in the opportunity for profit gain. According to Rouibah and Al-Rafee (2009), assessment of end-users needs is one of critical determinants of success in an information system project. Dorfman (1999) reports that value of good software requirements and the need to do them well, increases as the software size and complexity increases. However, the challenge is to identify software requirements and determine how they can be met. Rouibah and Al-Rafee (2009) report about failure of software projects in Kuwait primarily due to wrong, incomplete or misunderstood user requirements. Hofmann and Lehner (2001:58) posit that the biggest cause of failure in software projects is due to deficient requirements. According to Dorfman (1999), deficiencies in requirements are among the prime contributors to problems related to failure in software projects. Similarly, Jiang, Eberlein, Far and Mousavi (2008:303) report that the quality of a software product is determined by the requirements engineering process involved. Therefore requirement analysis of a software project; which is a process of identifying and determining requirements to be supported by a software system to be developed, is critical for the success of a given software projects.

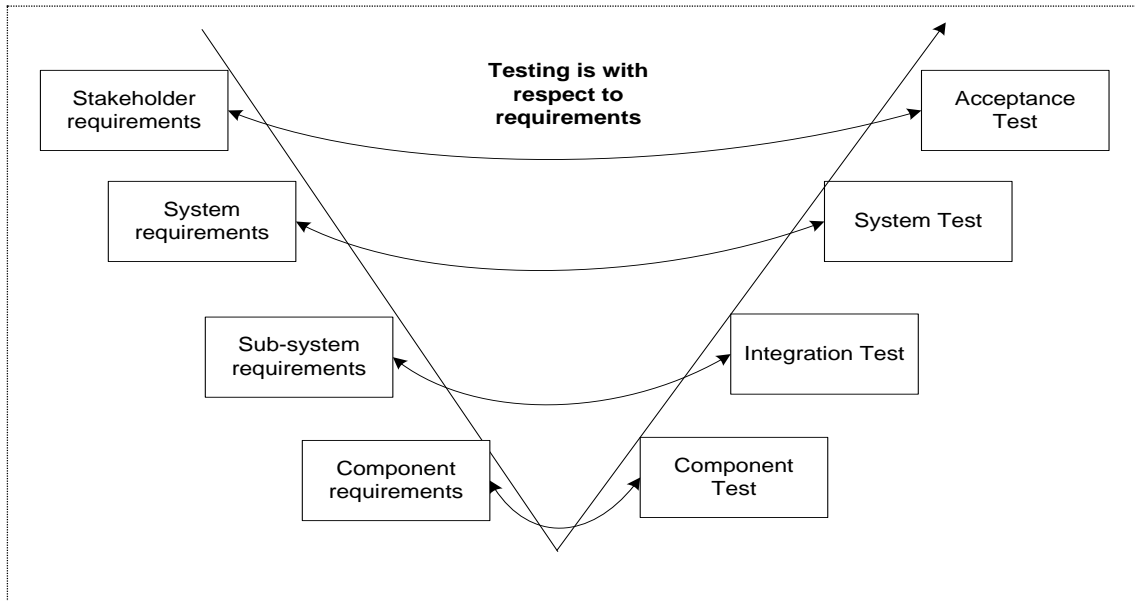
Requirement analysis is performed by identifying, determining and establishing user needs for the organization to meet its business objectives. According to Hofmann and Lehner (2001:59), requirements engineering is a process of specifying, analysing and refining requirements by studying stakeholders' needs. Stakeholders refers to people, such as customers, managers and end-users, who have interest in a given project and who in one way or the other will be affected by a software system to be developed (Hughes & Cotterell, 2006:13). Hofmann and Lehner (2001: 58) give examples of stakeholders as customers, end users, project managers, analysts, software developers, senior management and quality assurance staff. Furthermore, in their research, Hofmann and Lehner (2001:65) report that the most successful software project teams always involved customers and users during requirements engineering process and maintained a good relationship with stakeholders. Stary (2002:425) mentions that tasks performed by users and user characteristics are key elements during the development of a software system. Moreover, Stary (2002:425) states that analysis phase aims towards acquisition and elicitation of requirements from users and the organization to which a software system is to be developed for.

2.4.1.1 Types of software requirements

According to Bosch and Molin (1999), software requirements can be categorised as functional requirements and non-functional requirements. Bosch and Molin (1999) explain that functional requirements are those requirements related to the domain functionality of the application. Moreover, Bosch and Molin (1999) further divide non-functional requirements into development non-functional requirements and operational non-functional requirements. Development non-functional requirements refer to qualities of the system relevant to software engineering such as reusability, flexibility and maintainability while operational non-functional requirements refer to qualities of the system in operation such as performance, reliability, robustness and fault-tolerance. Non-functional requirements are also known as quality requirements.

Sommerville (2007:119) posits that software requirements can be classified into three groups: functional requirements, non-functional requirements and domain requirements. Functional requirements describe what the system is expected to do in order for users to execute their tasks. Non-functional requirements refer to how well the system performs what is supposed to do. This includes system characteristics such as system performance, reliability, usability, availability, security and others. These are emergent properties of a system since they can be determined once the

system is completed. Domain requirements are system requirements pertaining to the application domain of the system to be developed. Domain requirements are



business specific, for instance a standard user interface in a particular domain.

Figure 2.1: Requirements in the V-model (adapted from: Hull et al, 2005)

With reference to figure 2.1, Hull, Jackson and Dick (2005) use the V-model to categorize four layers of requirements as stakeholder requirements, system requirements, subsystem requirements and component requirements. This is rather the perspective or levels through which requirements can be examined, tested and confirmed. Stakeholder requirements refer to a high level of requirements where users or stakeholders are mainly concerned with needs to accomplish business tasks. System requirements refer to a level where stakeholder requirements are translated into needs that the system must facilitate for users to achieving business goals. Sub-system requirements refers to requirements that should be achieved by a sub-system while component requirements is a lower level of requirements where concerns are on what a particular component can achieve in order for the system to perform its tasks.

Requirements can be verified and validated to have been met during testing phase. Hull et al (2005) mentioned that the V-model can be used to test and verify whether requirements were met. This can be done by performing component test, integration test, system test and user acceptance test versus component requirements, sub-

system requirements, system requirements and stakeholder requirements respectively (figure 2.1).

2.4.1.2 Requirements life cycle

According to Hofmann and Lehner (2001:59) requirements engineering includes four separate but related activities: elicitation, modelling, validation and verification. Jiang *et al.* (2008) categorize requirements engineering process into requirements elicitation, requirements analysis and negotiation, requirements documentation and requirements validation. With reference to Figure 2.2, Sommerville (2007:143) reports that requirements engineering can be divided into four high-level sub-processes: feasibility study, requirements elicitation and analysis, requirements specification and requirements validation as follows:-

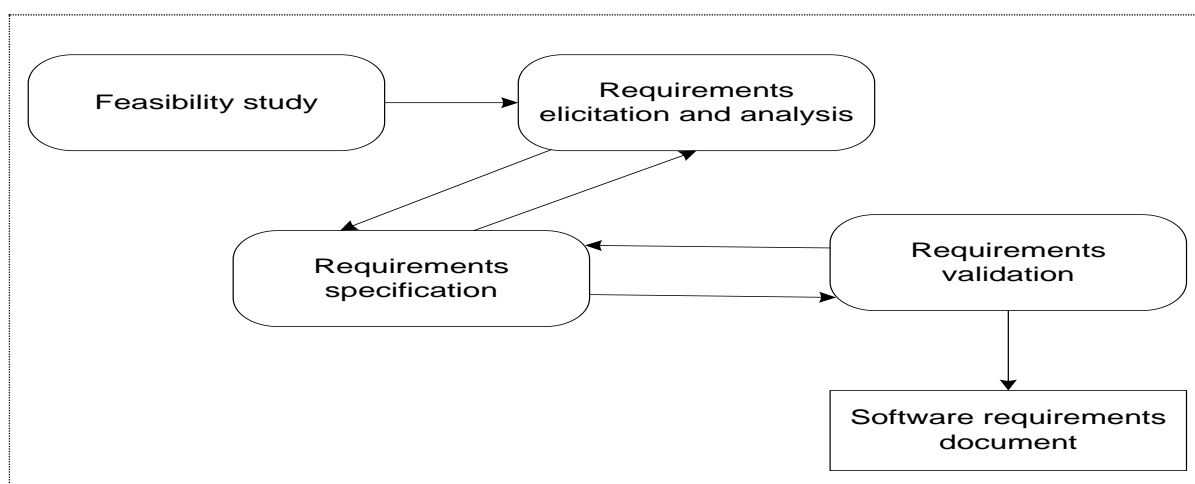


Figure 2.2: Requirements engineering process (adapted from: Sommerville, 2007:143)

- i. *Feasibility study is concerned with determining the business case of the system; whether the system to be developed will be useful and will enable the organization achieve its business objectives.*
- ii. *Requirements elicitation and analysis involves processes that will facilitate the discovery of requirements for the system to be developed.*
- iii. *Requirement specification refers to the process of converting discovered requirements into some standard forms or documents. A document produced during this process is called software requirements specification (SRS).*

According to Sommerville (2007:136), it is the SRS document that software developers should implement during software development activities.

- iv. *Requirements validation is concerned with checking to ensure that user requirements reflect what customers want in order for the system to perform what is expected for the organization to achieve its business goals.*

Elicitation, specification and validation processes of requirements are iterative in nature. This is due to the fact that requirement engineering is a discovery process where more and more requirements appear and becomes clear as the software development process continues. With reference to Figure 2.2, requirements discovered during elicitation should be documented during specification process. The documented requirements will then be validated during the validation process. However during validation other new requirements can lead to the discovery of other requirements. Also during specification process new requirements can be discovered hence leading to the iterative nature of execution where some requirements lead to the discovery of other requirements as indicated by arrows in Figure 2.2.

2.4.1.3 Techniques used during requirements analysis

There are many techniques and tools that traditionally have been used to facilitate the requirements analysis process. Stary (2002:437) reports that, while performing requirement analysis, interacting with users in their actual work environment is vital to designing usable software products. Furthermore, Stary (2002:437) mentions that user and task analysis is an analysis technique that can facilitate the design of a product that can change the culture observed among users. This is especially important if cultures observed makes users less productive while performing their tasks. Sommerville (2007:152-158) and Jiang *et al.* (2008:326) describe about interviews, use cases, prototype and ethnography³ as techniques used to facilitate requirements analysis. Olson (2004:93) reports about meetings, interviews and brainstorming as other techniques used to elicit user requirements.

³ Ethnography is an observational technique used to understand social and organisational requirements where an analyst observes the day-to-day work and notes made of the actual tasks in which participants are involved (Sommerville (2007:157))

2.4.1.4 Skills required during requirement analysis

All techniques used during requirements analysis involve meeting among software development team-members and stakeholders. According to Hofmann and Lehner (2001:65), the most successful project teams involve customers and users and maintain good relationship with stakeholders. Hence understanding user requirements require that software developers and stakeholders continuously work together, starting from early stages of a software project.

To be proficient in requirement analysis one should be a good communicator, good listener and negotiator (Hughes & Cotterell, 2006:13). According to Olson (2004:29), good communication is a major factor for successful software projects. The requirements analysis phase requires intensive communications and frequent interactions among software developers and customers. Hence, communication skills are vital during requirement analysis.

2.4.2 System and software design

According to Bennett *et al.* (2006:50) an information system involves hardware, software and people as main components. Sommerville (2007:67) reports that system design is concerned with determining and grouping requirements into requirements that can be addressed by hardware and requirements that can be addressed by software. Bosch and Molin (1999) define system requirements as a top-level requirement set composed of software, hardware and mechanical requirements. System design aims at establishing the overall system architecture, which includes hardware, software and people. On the other hand, software design, according to Sommerville (2007:67), involves identifying and describing the fundamental software system abstractions and their relationships. Sommerville (2007:242) mentions that architectural design involves identifying sub-systems, establishing a framework for sub-system control and the communication process among sub-systems.

Stary (2002:425) reports that the objective of design is to specify the structure and behaviour of the software as it should be implemented. Bennett *et al.* (2006:340) define software architecture as the organization of a system in terms of system components or sub-systems and the manner in which those components or sub-

systems communicate to each other. Bosch and Molin (1999) mention about functionality-based architecture design whose main objective is to identify the core abstractions, model abstractions as objects and determine interactions among the identified system abstractions or objects.

The goal of software design is therefore to establish software architecture which is a blueprint depicting the software to be developed and implemented by software developers. Software developers are more involved with software design activities while system architects are more involved with system design activities. According to Malan and Bredemeyer (2002:11), good software architecture should have three major characteristics; namely good, right and successful, as mentioned below:

- *Good: it is technically sound and clearly represented*
- *Right: it meets the need and objectives of key stakeholders*
- *Successful: it is actually used in developing systems that deliver strategic advantage*

As such, it is the responsibility of software developers to ensure the design of good, right and successful software architecture. It remains important to recognize that the goal of software design is therefore to identify system components that make up the system and establish how those components will communicate or interact to one another.

2.4.2.1 Advantages of software architecture

According to Sommerville (2007:242), software architecture can be used as a means of communication among stakeholders. With reference to Malan and Bredemeyer (2002:10), software architecture is a central thinking and communicating tool for architects and the development team. End-users, software developers and management have to communicate to one another in order to ensure that functional and non-functional requirements of a software system are met.

Stary (2002:426) reports about a design concept called “user-centred design” in which knowledge and involvement of users is critical during the designing process hence the need for extensive communication among users and software developers. However, the nature of information to be communicated differs from technical to non-technical information. While technical information is understood and well accepted by software developers, it may create confusion among end-users and managers.

Hence designed system components and the software architecture as a whole can facilitate communication among both technical and non-technical stakeholders.

Due to the complexity nature of software systems, software architecture makes a complex software system more understandable and intellectually manageable (Malan & Bredemeyer, 2002:2). This is enforced by the design of system abstractions, which hide unnecessary details and decompose complex system into smaller and manageable components; commonly known as the principle of divide and conquer (Malan & Bredemeyer, 2002:4-5).

Bosch and Molin (1999) mention about the risk of wasting resources; where resources are put on developing a system, and only to realize that the system does not meet user requirements; the situation that can make a software system not usable and hence abandoned by users. Software architecture can be used to determine whether critical requirements will be met.

According to Reeves (1992), programming is not about building software; programming is about designing software. The main objective of software design is to ensure that user requirements both functional and non-functional requirements are met by the system to be developed. Software design can thus suggest early changes to the system before the system is developed. According Hughes and Cotterell (2006), late changes to software are more expensive than early changes. Hence, software design makes it cheaper to develop software in terms of time, budget and the limited resources.

Moreover, Sommerville (2007:242) reports that system components designed during software design can be re-used to develop software systems to address similar user requirements. Software design supports software re-use and therefore it can speed up the software development process.

2.4.2.2 Categories of design

Design as a process of converting user requirements into a blueprint that will be used to develop a software system, can be viewed as either general design or specific design. Bennett *et al.* (2006:376) posit that design can be grouped into two levels; system design and detail design. Bennett *et al.* (2006:376) describe system design as a design concerned with the overall and high-level architecture of the system

while detailed design is concerned with designing of low level and detailed individual software components. Good software design is required at all levels of design. Sommerville (2007:247) refers to system design as system organization, which is a strategy used to structure the overall system.

Sommerville (2007:247-252) describes common types of system design as follows:

- i. The repository model: refers to a structure where components involved communicate to one another by sharing information. The shared information is usually stored in the centralized database hence the name repository. However, each component can store its data and only pass that data when required by another component.*
- ii. The client-server model: refer to the design structure where components are grouped into client and server components. Client components include those components that use services while server components are those components that provide services. The essence of client-server model is based on grouping component into sets of services and determining components that deliver services and components that access and consume services.*
- iii. The layered model: refers to system design where system services are grouped into sets of services and each set of services is provided by a particular layer. The essence of this model is determining services to be provided by the system, determining number of layers required and allocating services to appropriate layers. According to Sommerville (2007:251), the layered approach supports the incremental approach to system development. Hence, services of developed layers can be made available to users immediately.*

Bennett *et al.* (2006:350-356) explain that on the layered architecture of software systems, applications are developed as separate layers of services. With reference to Figure 2.3, the most common layered architecture used for software development is a three-tier architecture, which is comprised of presentation layer, logic layer and data layer (Bennett *et al.*, 2006:353).

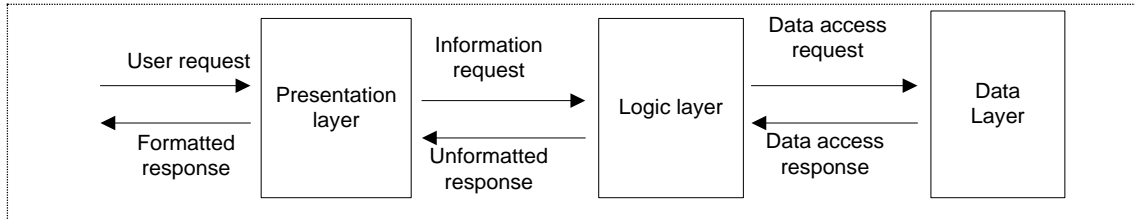


Figure 2.3: The three-tier architecture (adapted from Satzinger *et al.*, 2004:334).

Presentation layer deals with how the user will interact with the system, view data and how the program will present data to the user. Logic layer is concerned with the business rules that should be implemented in order for system to achieve its functions and for users to be able to use the system and achieve their business goals. Data layer deals with data management and data storage.

Among others, the main advantage of the layered architecture is that it makes the system manageable and maintainable. This is important because most of the software development work is maintenance (Sommerville, 2007:67; Hughes & Cotterell, 2006:7). Therefore, it is vital that software systems are developed with future maintenance in mind and the layered architecture is engineered to support maintenance, reusability, reliability, flexibility, and other quality requirements.

Sommerville (2007:744-767) reports on another software design architecture; Service Oriented Architecture (SOA) which is a software design that makes development of large and complex software systems manageable. SOA is a significant development in the field of software development. Thus, the following section discusses SOA; a promising design architecture for software systems.

2.4.2.3 Service oriented architecture

One of many challenges in software development is to deal with volatile and dynamic nature of user requirements. Following the dynamic nature of business requirements and complexity nature of software development, a software product may take longer to develop and by the time the product reaches the customer, business requirements may have changed (Beck, 2005). This situation can lead to a software system being abandoned by its users since it is no longer usable and hence serve little or no purpose from the user perspective. Software developers must have the ability to deal with volatility and dynamic nature of business requirements in order to ensure that users' objectives are always satisfied. Service Oriented Architecture (SOA) is a type

of software design geared towards addressing volatility and dynamic nature of user requirements. According to Lowy (2007), Service-orientation is the correct way to build maintainable, robust, and secure software applications.

SOA is a software design architecture, which considers business applications as collections of various functions or services that provide and support particular user requirements (Ort, 2005). Instead of developing a software system from scratch, SOA advocates software reusability by integrating services that already have been developed. SOA is an information technology strategy of making use of services available in a network such as the Internet. SOA is an agile alternative to address and respond to ever-changing business needs for customer satisfaction (Bennett *et al.*, 2000). Services are made to interact and communicate to one another in order to exchange information and serve specific user requirements (Ort, 2005). A software component that provides a particular related group of functions can be referred to as a service.

Web services is a service oriented architecture (SOA) that provide means to speed up the software development process and building systems that are adaptable. According to Sotomayor (2005), web services are platform independent and language independent because they use standard eXtensible Markup Languages (XML). A client program can be developed in a particular programming language and be deployed in a particular platform different to a web service's programming language and deployment environment. For instance, a client program can be developed using C-sharp (C#) as a programming language and be deployed in Windows platform. However, the web service that provides services to the client program can be developed using Java as a programming language and can be deployed in Linux platform.

The following section explores about the web service architecture; an important area in the field of software development. Web service technology is increasingly applied in addressing ever-changing user requirements and development of large and complex business applications (Sommerville, 2007:744). Knowledge of web services is critical among software developers. Attention is given to the fundamental characteristics and main building blocks of web services. The mechanism as to how web services operate in order to provide services to client programs is explained and advantages and disadvantages of using web services are discussed.

2.4.2.3.1 Web services

The web technology is expanding and growing in order to enable more sophisticated forms of interactions between client⁴ applications and server⁵ applications. The web is a giant distributed content library (Curbera, Nagy & Weerawarana, 2001). The evolution of the web has resulted into web services; the technology that enhances complex business-to-business (B2B) interactions. In this kind of interactions a program communicate to another program (program-to-program interaction) through a well-defined interface. In order to enable interaction, integration and interoperability between applications, web services support the dynamic nature of the web by supporting just-in-time application integration.

A web service is a container and is composed of a collection of operations or functions. These functions can be accessed on the network through the exchange of XML formatted messages or JavaScript Object Notation (JSON) formatted messages between a client application and a server application. Pras and Flatin (2007) report about two groups of web services as fine-grained web services and coarse-grained web services. Fine-grained web services are web services specialized to perform a specific task while coarse-grained web services are web services that perform a specific group of related tasks.

Web services support component-oriented software development where applications are encapsulated and only interact through well-defined interfaces. Hence web services are called “gray box” components because they are encapsulated programs that are described by their description files (Curbera *et al*, 2001). The description file is the interface with which applications use in order to communicate to one another. Web services obey the loosely coupled interaction model, which allows scalability in applications, since applications are independent on one another; change in one application should not necessarily require changes to other applications. This is essential because Web services are fundamentally language-free and platform-free (Sotomayor, 2005). Web services allow for flexible integration and the level of integration can be extended to the level supported by common and efficient protocol (Curbera *et al*, 2001). Web services use message and document format as a basis

⁴ Client: an application that request services

⁵ Server: an application that provide services

for describing services and for industry standard instead of Application Programming Interfaces (API) in order to ensure service interoperability (Curbera *et al*, 2001)

According to Beznosov, Flinn, Kawamoto and Hartman (2005), a web service is a distributed computing oriented technology build on the foundation of other previous distributed computing technologies such as Distributed Component Object Model (DCOM), Common Object Request Broker Architecture (CORBA) and Enterprise Java Beans (EJB). While these technologies have important roles to play, their interoperability is the most difficult part; applications developed in different environments can hardly communicate and interact to one another. Software applications developed by such technologies are language dependent and platform dependent. On the contrary, web services are language independent and platform independent; web services technology aims at making distributed computing easier and enhancing system interactions, integration and interoperability.

Gottschalk, Graham, Kreger and Snell (2002) report about business-to-consumer (B2C) interactions and business-to-business (B2B) interactions as the two major categories of software interactions on the Internet. Web services support and facilitate B2B interactions where programs communicate to each other. Just like websites are being used to publish information understandable by humans, web services are used to publish information that is comprehensible to other software systems (Sotomayor, 2005). According to Srivastava and Koehler (2003), a web service is an independent application logic that provides services or business functionalities to other applications through the Internet technology. In order to develop software that are language-free and platform-free, the building blocks of web services conform to World Wide Web (WWW) standard.

With reference to Figure 2.4, in order to enforce the interoperability values offered by web services, Sotomayor (2005) describes a four-layered architecture of web services, which consists of processes layer, description layer, invocation layer and transport layer. These four layers are described in the following section.

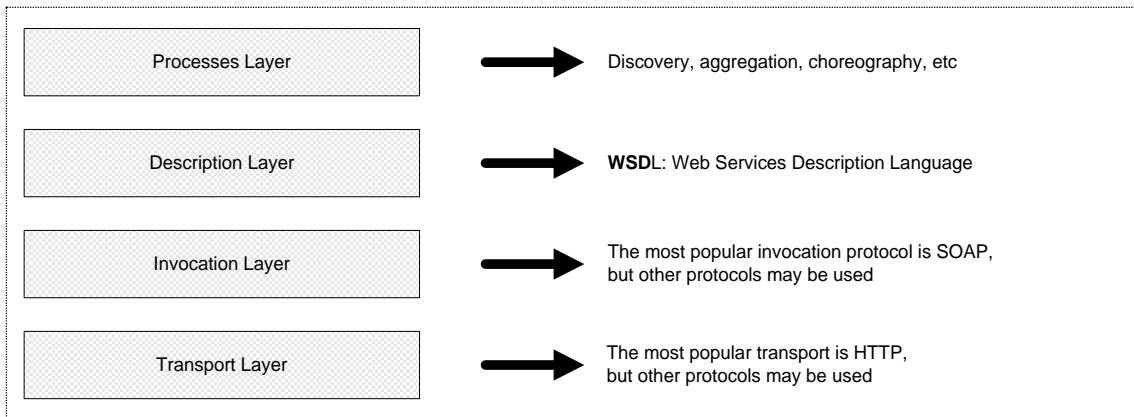


Figure 2.4: Four layers of web service architecture (adapted from Sotomayor, 2005)

i. Processes layer:

The processes layer facilitates discovery of web services so that client applications can find and locate web services with ease. The main technology used to locate web services is Universal Description, Discovery and Integration (UDDI). UDDI is a mechanism used in order to discover locations where specific web services are provided and gives information about the web service provider.

ii. Description layer:

In order for client applications to use operations provided by the web services, clients should know which operations are available and how to invoke those operations. The description layer allows the web service to describe itself in order to indicate operations supported and how those operations can be utilized or invoked by client applications. The main technology used to support self-description and easy invocation of web services is the Web Service Description Language (WSDL).

iii. Invocation layer:

The invocation of operations provided by a web service, involves back and forth exchange of messages between a client application and a server where a web service is located. However the messages communicated should be in a format understandable to both client and server. Hence the function of the invocation layer, which is to format messages prior transfer between client and server. The main technology used to format messages is called Simple Object Access Protocol (SOAP).

iv. Transport layer:

Once in the required format, messages will be transmitted between the server and client. Hence the Transport layer, which is a layer responsible for transmitting messages between client and server. The most used protocol to ensure message transfer in web services is called HyperText Transfer Protocol (HTTP).

With reference to figure 2.5, the three basic building blocks of web services are Simple Object Access Protocol (SOAP), Universal Description, Discovery and Integration (UDDI) and Web Services Description Language (WSDL). These building blocks of web services are defined using eXtensible Markup Language (XML) and other Internet protocols such as HyperText Transfer Protocol (HTTP). The standard of web service architecture is specified by World Wide Web Consortium (W3C), which is also responsible for the standard of eXtensible Mark Language (XML), HyperText Markup Language (HTML), Cascading Style Sheet (CSS) and other web related technologies (Sotomayor, 2005).

2.4.2.3.2 Communication mechanism in web services

With reference to figure 2.5, the communication process of web services involves two key role players; a service provider (server application) and a service requestor (client application). There are two parts involved in the communication process; service creation part and service consumption part. In the service creation part, a service provider designs and develops web services using WSDL and makes web service operations available to the service requestor. In order to makes the operations of web services available, the service provider have to publish the address of web services, also known as Uniform Resource Identifier (URI), to a service registry by using the UDDI technology (Gottschalk *et al.*, 2002). Once the URI has been published, the service is available and can be consumed by authorized service requestors.

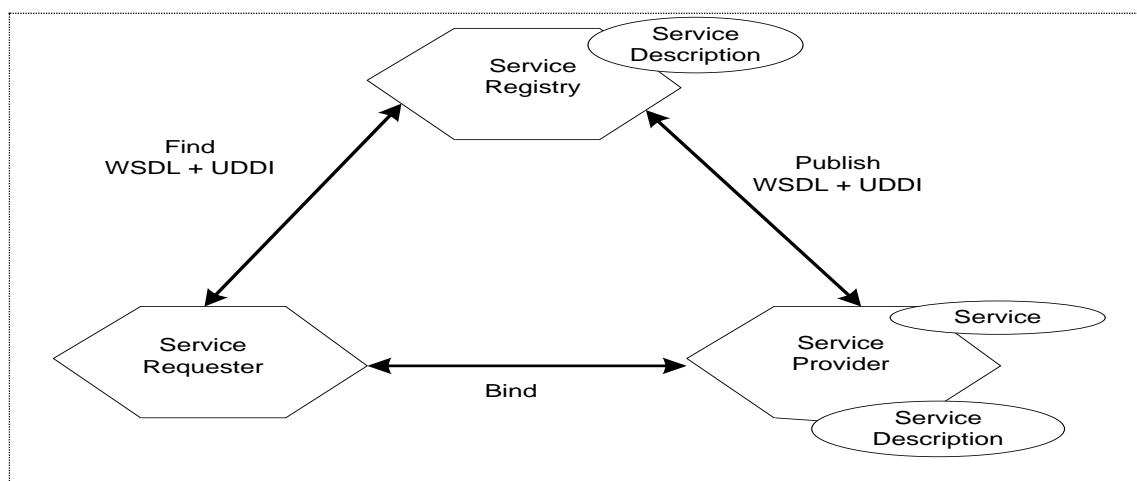


Figure 2.5: Web service objects and operations (adapted from Gottschalk *et al.*, 2002)

During service consumption, service requestor utilizes services provided by the service provider. In order to use a web service, the service requestor has to find the URI of the web service offering the requested services in the service registry. The service requestor then uses the URI to retrieve and attach itself to the web service in order to use the required web service operations. The process of attaching a service requestor to a web service is called binding (Sotomayor, 2005). When the service requestor discovers the location of a web service, it creates a request message in a special format using SOAP or JSON. Then the request message is transmitted through the Internet protocol, mostly HTTP, to the appropriate web service. The web service executes its operations, which results into a response message. Then the response message will be formatted using SOAP or JSON and transferred back to the appropriate service requestor via HTTP protocol. The service requestor does not directly interact with the web service, the communication and interaction between service requestor and web service is made possible through a proxy class (Wang, Huang, Qu & Xie, 2004). Once the service requestor receives response from the service provider, the communication process ends. The interaction between service requestor and service provider disbands and the communication process stops.

Complex business processes require multiple communication processes to be performed by several web services. This is achieved by specific invocations and interactions of multiple web services through a coordinated process called orchestration or choreography (McGibbon, 2007). This orchestration process is achieved by using an XML-based language called Web Services Business Process Execution Language (WS-BPEL).

2.4.2.3.3 The importance of web services in business

The dynamic nature of business requirements is a challenge to software development. This challenge can be addressed by using business applications developed at a fly to meet immediate and ever changing business needs (Wang *et al.*, 2004). The web services architecture promises to address this problem of constantly changing business requirements. Hence, the web service is ideal for e-Commerce where business requirements are dynamic and volatile. Web services capabilities of supporting interoperability among applications, scalability, flexibility, loose coupling, reusability, language independence and platform independence makes it ideal for ever-changing business requirements (McGibbon, 2007).

Web services allow business organizations to utilize the Internet as a common platform for communication, interaction and exchange of data in order to carry out business activities and provide valuable services. Web services can facilitate both business-to-consumer (B2C) and business-to-business (B2B) interactions among business organizations. Despite concerns about security and trust issues related to using Web services from external service providers, merits of using web services by far outweigh demerits. Knight (2005) reports that service oriented architecture (SOA) will dominate design and development of new software products. Therefore, competency in developing services oriented software products is critical to software developers. This is because many business organizations are increasingly utilizing web services to conduct their business transactions.

The following section discusses user interface design as a critical phase during system and software design phase.

2.4.2.3.4 User interface design

Software systems are increasingly becoming part of human life and social activities (Kofman, 2006). Software users issue instructions and receive feedback from computerized programs via the interface also known as Human Computer Interface (HCI). Good communication and interaction among users and software systems can be determined by HCI. The design of user-friendly HCI is critical for usable and dependable software systems (Sommerville, 2007:363). According to Sommerville (2007), software developers often develop user interfaces that are inappropriate and difficult to use by the intended audience. In order to address this problem Bennett *et al.* (2006), among others, recommend the use of “the user-centred approach” where

users are involved in every aspect of interface design. It is by involving users, that software developers will be able to design and develop attractive, appropriate, friendly, usable and dependable software applications.

According to Bennett *et al.* (2006), the attitude of users towards using the system can be influenced by their experience with the user interface of a given system. Users will opt to using programs that are user friendly, intuitive and simple to follow (Bennett *et al.*, 2006). It is therefore important that software developers think about users and apply the user-centred approach in order to produce usable software systems.

The following section 2.3.2.5 discusses about object orientation as a critical phase during system and software design phase.

2.4.2.3.5 Object Oriented Programming

The concept of Object Oriented Programming (OOP) is based on the premise that people view their environment as a collection of objects (Satzinger *et al.*, 2004). According to Kendal (2011:20), the perception of software system as a collection of objects interacting together to achieve a particular goal, is a more natural way of looking at a software system to be developed. Kendal (2011:113-122) further suggests that users think in terms of objects as such their requirements can be correctly captured if viewed as collection of objects interacting one another. In order to undertake software development activities, it is suggested that things involved in the software development requirements be categorized into groups or abstractions. The grouping should be done in such a way that things of similar characteristics are grouped together. Hence the term “class” which literally refers to a blue print which can be used to create instances of classes, also known as objects (Bennett *et al.*, 2006). The aim of designing classes is to analyse and identify objects needed by the software system to execute its activities.

A class has particular characteristics that distinguishes it from other classes and supports methods or operations that could be used to access its characteristics (Malik, 2002). A class is said to be self-contained; it has data and operations encapsulated in one container. Different sets of classes are used by the software system to created different sets of objects required by the system, the process known as instantiation of objects (Bennett *et al.*, 2006:71).

Programming codes proven to perform effectively and efficiently can be encapsulated into classes and reused in other similar software projects. This is software reusability. Software reusability is a powerful technique because if used appropriately, it can speed up the software development process. In addition, software reusability can prevent a software problem of “inventing the wheel” where all programming codes are developed from the scratch and hence reduce production costs (Sajeev, 1994:1161).

Most of software developer’s tasks are maintenance of previous software codes. Maintenance can be difficult and tedious work in large and complex systems. Classes however make maintenance work manageable. Classes alleviate the problem of writing same piece of code in different areas of the software. In large software systems, it can be tedious to manually modify codes in all areas affected by changes of a given functionality. Nevertheless, classes comes to rescue because modifications of codes can be performed in one specific area of a class and changes propagated to other classes via inheritance and composition, as explained below.

Inheritance refers to characteristics of one category of things being passed to all things that belong to the same category. This is data abstraction where common characteristics of things are grouped together and passed to other things of the same category. This prevents duplication of codes because things with similar characteristics are grouped in one category and those characteristics can be passed to other things that require similar properties by a process called inheritance (Satzinger *et al.*, (2004:183). Inheritance involves a super-class; a class that contains all common characteristics and sub-class; a class that inherits all properties from super-class. It is also known as generalization⁶ and specialization⁷ of classes and it caters for “is a” relationship” (Bennett et al., 2006). For instance, a student is a person and a teacher is a person. This implies that a person is a super-class with all characteristics that describes a person. On the other hand, a student and a teacher are sub-classes and inherit all properties of a person.

Composition and aggregation describe another form of relationships among objects interacting in the software system. This is the “has a” relationship. For instance, a

⁶ Generation is hierarchical relationship from sub-class to super-class

⁷ Specialization is hierarchical relationship from super-class to sub-class
(Deitel & Deitel, 2006: 549)

computer has a keyboard and mouse; however, neither a keyboard nor a mouse is a computer. The distinction between composition and aggregation is based on the degree of dependency relationship between two interacting objects; composition represent stronger dependency relationship while aggregation implies weaker dependency relationship (Bennett et al., 2006). For instance, a computer can exist without a mouse but cannot exist without a RAM (Random Access Memory). Therefore, computer has a composition relationship with RAM and aggregation relationship with a mouse.

Another unique concept related to the OOP approach is “information hiding”. This refers to the ability of objects to have data and operations self-contained (data encapsulation) which results into an object knowing only about itself. An object is not concerned about how other objects perform their functions. According to Wirfs-Brock (2009:11), hiding implementation details preserves design flexibility. Details of the mechanisms and processes required to perform a particular work in one object, are hidden from other objects. If one object requires a particular function to be carried out by another object; it does so by sending a message to another object; how the operations are done is not a concern of the object but of the other object receiving the message (Malik, 2002).

The OOP approach addresses complexity associated with the development of software systems. This includes maintenance of software products, scalability of systems and reusability of software components. Knowledge of OOP is significant in the software development industry.

2.4.2.3.6 Techniques used during software design

Bennett *et al.* (2006:415-437) explain that design patterns are recurring solutions to software design problems. A typical design problem is component mismatch when designed components of a system do not fit to allow thorough components interactions. Garlan *et al.* (2009:68) report that design patterns such as wrappers, façade, adapters, mediators and bridge can resolve component mismatch problems. As such, knowledge of design patterns is critical in resolving software design problems.

A modelling language called UML (Unified Modelling Language) is commonly used to design models of software systems. According to Bennett *et al.* (2006:103-126),

UML is particularly used to design models such as sequence diagrams, interaction diagrams and activity diagrams, in order to illustrate the interactions of objects in a given business scenario. Among others UML is also used to model use case diagrams in order to demonstrate interactions between users and a given software system. Knowledge of UML and object orientation is significant during the design of large and complex software systems.

While designing a software product, a software developer has to correctly interpret specifications and convert those specifications into design models. In the structured paradigm of programming; where a software program is viewed as a collection of functions, software developers use techniques such as system flow charts, structure charts and pseudo-code (Satzinger *et al.*, 2004:348-367). On the other hand, in the OOP approach where a software program is viewed as a collection of interacting objects, software developers utilize use case diagrams to identify and determine required functionalities of a system (Bennett *et al.*, 2006:145-154).

According to Sommerville (2007:326-335) techniques used during software design include class diagrams, communication diagrams, sequence diagrams and state diagrams. Bennett *et al.* (2006:397-412) explain that a class diagram is composed of a collection of classes and their relationships. Bennett *et al.* (2006:252-277) indicate that sequence diagrams and communication diagrams are important techniques for modelling interactions among objects in a given system. Furthermore, according to Sommerville (2007:329) state machine diagrams are useful in showing how individual objects change their state in response to events.

2.4.3 Implementation and software testing

During implementation and software testing, software design is converted into sets of program units (Sommerville, 2007:67). These program units include executable programming codes such as classes and functions. During this phase, software design components are converted into programming codes. In order to ensure that user requirements are met, testing is performed for each component developed. This process is known as unit testing where each piece of code written is individually tested to ensure that it respond appropriately to test cases and functional requirements (Sommerville, 2007:547).

According to Godefroid, Halleux, Nori, Rajamani, Schulte, Tillmann and Levin (2008:30), unit testing is a popular way to ensure early and frequent testing while developing software components. Unit testing is primarily performed by software developers since software developers are familiar with components that are being developed. Software developers can easily pinpoint errors and correct software defects, also known as bugs (Sommerville, 2007:547). Tillmann and Schulte (2006:38) report about software projects at Microsoft where software developers wrote more lines of code for unit testing than for the implementation being tested. When software developers know that they are required to test their specification and code, they design their code for testability which makes hard-to-test features testable (Talby, Hazzan, Dubinsky & Keren, 2006:32).

According to Sommerville (2007:547), the main objective of unit testing is to expose faults, defects or bugs in software components. Bugs are bad codes that affect the normal functioning of software systems. Bugs can cause security vulnerabilities such as buffer overflows and memory leak, which can be exploited by malicious attackers (Godefroid *et al.*, 2008:32). This leads to the debugging process, which is a process of removing bugs from software components. According to Hailpern and Santhanam (2002:5), the purpose of debugging is to locate and fix the offending code responsible for violating known specifications.

The ultimate product of implementation and software testing phase is defect-free executable software components that integrate to form a software system. According to Hailpern and Santhanam (2002:4), testing is a very expensive process that can cost from 50 percent to 75 percent of total development costs. In addition, Bertolino (2007) agrees that testing can consume 50 percent or more of total development costs. Nevertheless, it is very important that implementation and unit testing is performed thoroughly because the price of poor testing is costly and can be detrimental especially to safety and critical software systems.

Techniques and skills used during implementation and software testing are explained below.

There are tools that can assist software developers with generating and testing codes, however, writing and testing codes is performed by software developers. Bennett *et al.* (2006:563) mention about two categories of testing techniques as white

box testing and black box testing. White box testing refers to a process of examining specifications and programming codes in order to determine defects to be corrected. This includes stepping through codes line by line for each piece of code. White box is also known as structural testing since it involves checking the structure of specifications and programming codes (Sommerville, 2007:557). On the contrary black box testing requires no knowledge of programming codes; inputs and outputs of test cases are prepared, a program is executed with inputs and then a check is performed to assert that only expected outputs are produced. According to Hailpern and Santhanam (2002:7) black box testing is based on external specifications and does not involve the understanding of the detailed code implementations.

Since the implementation and testing phase focuses on writing and testing executable programs, proficiency in appropriate programming languages is critical. Common mainstream programming languages used to write software codes are Java, PHP, C-Sharp, C++, Perl and Python.

According to Satzinger *et al.* (2004:487-524) many software systems require data to be created, stored, retrieve, modified and deleted. As such knowledge of database management system (DBMS) such as Oracle, DB2, Microsoft SQL server and MySQL is vital. DBMS manipulate data by executing programming codes or statements written using structured query languages (SQL). Therefore thorough knowledge of SQL is important especially in data driven software systems.

Experience in debugging techniques play important roles to ensure that bugs are limited since eliminating all bugs is practically not feasible (Hailpern & Santhanam, 2002:9). Fowler (1999:53) reports about a technique called “Refactoring” where software components are changed internally without changing the external behaviour of codes. Refactoring is a code optimization technique performed in order to improve quality properties of software such as performance, security and scalability. According to Fowler (1999:56) refactoring can help software developers to write codes that are easy to understand and easy to debug.

2.4.4 Integration and system testing

This phase focuses on integrating software components into sub-systems and then integrating sub-systems to form a complete software system. According to Copeland and Haemer (2000:42), integration should be performed frequently as a normal part

of every software developer's daily work; the process called continuous integration. With reference to the V-model on Figure 2.6, all testing are performed according to various testing plans where testing plans act as links between development and testing (Sommerville, 2007:520). Once unit testing is complete, software components are integrated together and sub-system integration test is conducted according to the sub-system integration test plan. Thereafter different sub-systems are integrated and system integration test is conducted according to system integration test plan. Finally, prior to making the system operational, acceptance testing which involve customers is conducted according to the acceptance test plan.

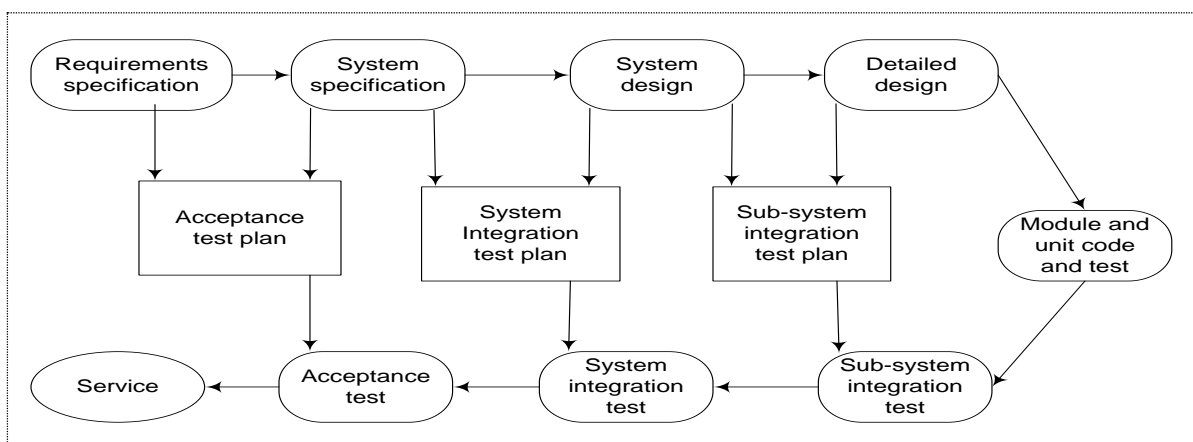


Figure 2.6: V-model showing test plans (adapted from Sommerville, 2007:520)

Integration testing is concerned with identifying defects in the system and ensuring that software components work together (Sommerville, 2007:540-541). Since integrating components could introduce new bugs or expose unidentified bugs, it is essential that software testing be conducted as components are integrated. This goes hand in hand with verification and validation (V&V) whose objective is to ensure that requirements specification and customers' expectations are met (Sommerville, 2007:516-520). With regard to requirements specification, the system developed should meet both functional and non-functional requirements. Moreover, with regard to meeting customers' expectation, the system should cater for business needs as expected by stakeholders. This leads to system testing where the system as a whole is tested to ensure that both functional and non-functional requirements are met. According to Hailpern and Santhanam (2002:7) system testing targets key aspects of software products, such as recovery, security, stress, performance, hardware configurations and software configurations. Before the system is made operational,

acceptance testing is conducted, where customers are involved in order to ensure that customers' expectations are met (Olson, 2004:245).

Techniques and skills used during system testing are explained below.

White box testing can be implemented during integration and system testing to examine codes that have been used to integrate components. However, black box testing is largely preferred since the objective is to understand how the system behaves and responds to user inputs, without checking the details of the implementation codes. Hailpern and Santhanam (2002:10) report that regression testing is commonly performed where testing is repeatedly done to ensure that changes made to a system after integrating components do not introduce bugs. This ensures that a software system continues to meet requirement specifications and delivers customers' expectations. Furthermore Juristo, Moreno, Vegas and Solari (2006:78) mention that regression-testing techniques are widely used for selecting test cases from an existing set.

Sommerville (2007:81) explains about alpha testing, beta testing and acceptance testing. Alpha testing is used where a particular single client is involved in the testing process as the system is being integrated and tested. Thereafter beta testing can be employed where software is released to multiple clients and exposed to the actual work environment using actual data. As the software is being used, reports are gathered and used to resolve defects reported by clients. Beta testing is similar to acceptance testing where users take part during testing and the system is exposed to actual work environment using actual data.

Integration and system testing involves two categories of stakeholders; technical and non-technical people. Technical skills such as proficiency in programming languages and SQL are critical. In large software companies, there are specialized teams dedicated to testing software systems, nevertheless, it is still the responsibility of software developers to test software systems.

2.4.5 Operation and maintenance

Sommerville, (2007:493) describes three types of software maintenance as maintenance to repair software faults, maintenance to adapt the software to a different operating environment and maintenance to add or modify the system's

functionality. Satzinger *et al.* (2004:660) describe software maintenance as modification of a software product after delivery to correct faults, improve performance or other attributes or adapt the product to a changed environment. Maintenance is the longest phase since it continues if the business organization is to address the ever-changing business needs.

Once acceptance testing is successful, the system is installed to users' work environment also known as deployment. Users start using the new system to accomplish their business tasks. At this stage, it is expected that the system provide all if not most of functional and non-functional requirements, as per requirements specification. However, following reasons such as continuous changes in business requirements to address changing business needs, maintenance will continue during the lifetime of business. Now that the system is exposed to the actual work environment where the workload may be high, new bugs could emerge. Moreover, hidden bugs that were not detected during previous project phases could materialize. Hence, the need for continuous maintenance where software developers continue to monitor the system, perform necessary adjustments and add more software functionalities as the business grows.

The actual development process might officially cease after deployment of the system, nevertheless, there are software developers who continue with the maintenance work. Most business organizations have software developers referred to as support team, available to assist users and perform necessary software maintenance.

Moreover since the system is new to users, training sessions are normally performed to users (Olson, 2004:244). When users understand how to effectively and efficiently utilize the new system, the system becomes usable. It serves no purpose if the system has all functionalities while users do not know how to utilize the system to perform their tasks. As such during maintenance phase, users receive necessary training regarding the functionalities of the new the system.

Techniques and skills used during operation and software maintenance are explained in the following section.

There are several techniques employed when the system is deployed to users. These techniques have different merits and demerits. Hence it depends on the nature of a software system, financial position of the business organization, work experience of users and the risks that a business organization is prepared to take. Olson (2004:245) mentions about parallel installation, the pilot operation approach and cold-turkey approach as common deployment techniques. Parallel installation refers to the software implementation process where both new and old systems are made operational at the same time. While this is reported to be a safer technique, it is costly since the organization has to run two systems hence there is an increase in the consumption of resources to support both systems. The pilot operation approach involves running the new system on limited basis. While this can help with identifying problems associated with the new system, workload related problem cannot be detected since the new system will be partially exposed to the work environment. Cold-turkey approach refers to the installation process where a new system is installed and an old system is completely removed from operation. The cold-turkey approach is sometimes known as the big bang approach. This approach can be risky since if something goes wrong with the new system, there is nothing to support business operations.

According to Bennett *et al.* (2006:568), there are four software installation strategies, namely, direct changeover which is equivalent to cold-turkey, parallel running, phased changeover and pilot project (Figure 2.7).

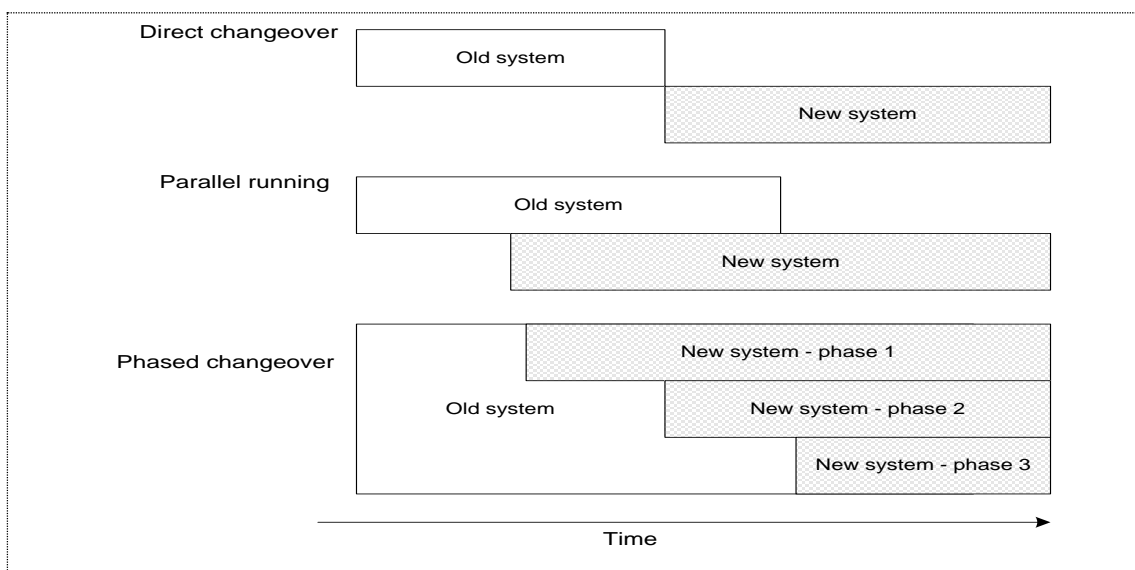


Figure 2.7: Software installation strategies (adapted from Bennett *et al.*, 2006:568)

The phased changeover approach refers to a technique where installation of a new system is performed not in one go but in phases. Another common technique is known as the geographical installation approach. This approach is useful when an organization is scattered in different geographical locations and hence installation of a new system is performed in one geographical location after another.

Software maintenance is also technical because it involves adding, modifying and removing programming codes in order to address business changes (Satzinger *et al.*, 2004:660). Thus proficiency in appropriate programming languages and SQL is important. Since hidden bugs could be discovered and exposed as the system is put to work, it is important that software developers have experience in debugging techniques. This is because one of the major roles during operation and maintenance is fixing bugs (Sommerville, 2007:493).

Finally, logical thinking and attention to details remain important skills during the operation and maintenance phase. Furthermore, a solid understanding of the whole software development life cycle from requirement specifications to operation and maintenance is critical. This is because from time to time software developers may have to re-visit previous phases in order to backward trace and understand sources of defects in a system in order to fix bugs.

2.5 Chapter summary

This chapter explored literature related to competencies in the software development field. The chapter commenced with an investigation on software projects followed by investigation of recent research pertaining to competencies. Finally, the chapter ended by a detailed discussion on the software development process.

2.6 Conclusion

Competencies for a given field are understood if and only if the philosophy and principles underpinning that field are understood. As such, the theoretical and practical understanding of the software development process is important in order to understand a software developer competency framework. After exploring literature on software projects, having investigated recent studies on competencies and the software development process, it is justifiable that knowledge of a software developer competency framework can improve competencies of software developers.

Therefore, a software developer competency framework is paramount for the success of software projects.

The following chapter three focuses on the research methodology.

CHAPTER THREE: RESEARCH METHODOLOGY

“Any fool can write code that a computer can understand. Good programmers write code that human can understand”

(Martin Fowler)

3.1 Introduction

This chapter discusses research methodology followed during this study. In particular, the theoretical framework underpinning this research is explained.

Research methodology for this study is geared towards understanding competencies required by software developers to perform their job as per the industry requirements. The chapter explores common research methodologies and typical research methodologies applicable in competency studies. The chapter further discusses about a research design followed in order to develop a Software Developer Competency Framework (SDCF).

The chapter commences with an investigation of categories of research paradigms in section 3.2. Section 3.3 discusses research methodology while section 3.4 discusses typical research methodologies used in recent studies on competencies. The research design for this research is described in section 3.5, followed by research population and sampling in section 3.6. Reliability and validity matters are discussed in section 3.7. Ethics, consent and confidentiality issues are discussed in section 3.8. The chapter concludes with chapter summary in section 3.9 and conclusion in section 3.10.

3.2 Categories of research paradigm

A study about a given phenomenon can be approached and attained from different perspectives commonly known as research paradigms. According to Mackenzie and Knipe (2006), a research paradigm is a theoretical framework that underpins a given research and influences the way knowledge is studied and interpreted. A research paradigm for a particular study is based on the ontological and epistemological views of the researcher. Ontology refers to the philosophical assumptions on how one view reality and being while epistemology refers to philosophical assumptions of how one

acquires knowledge (Mack, 2010:5). Ontology informs about epistemology, which then informs about methodology appropriate for a given study.

Bhattacharjee (2012:18-20) reports about four paradigms of social science research as functionalism, interpretivism, radical structuralism and radical humanism. These four categories of research paradigms, also known as four quadrants in research paradigms are based on the work of Burrell and Morgan (1979) regarding sociological paradigms and organisational analysis (Figure 3.1).

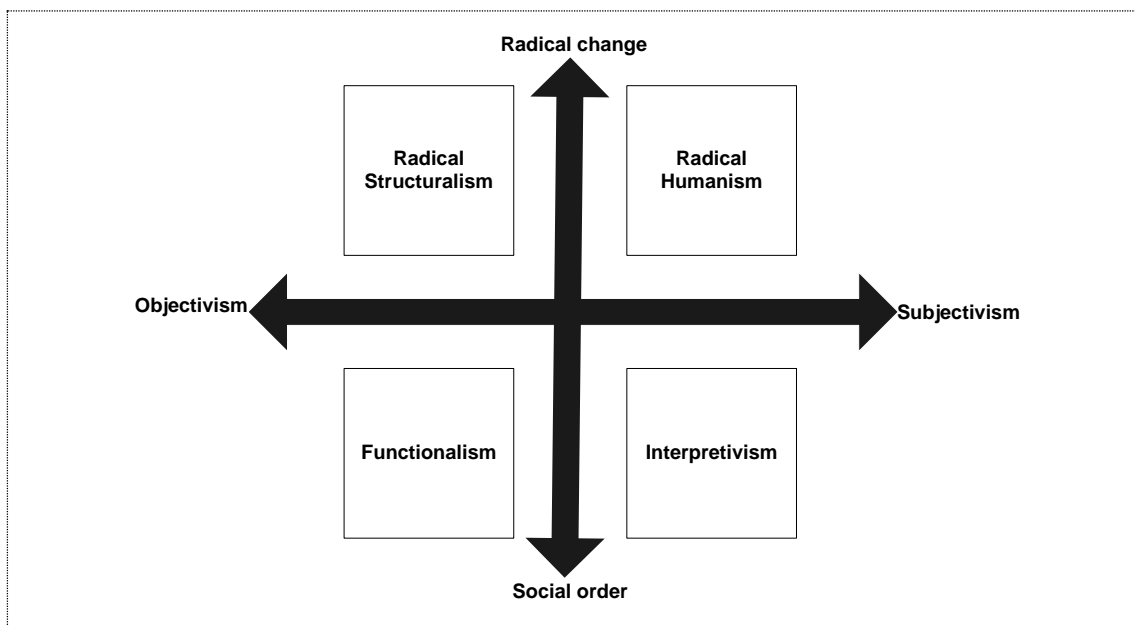


Figure 3.1: Four paradigms of social research (adapted from: Burrell & Morgan, 1979)

The difference of one paradigm to another is based on the ontological and epistemological assumptions or stance employed while conducting a research. Functionalists and interpretivists view the world as consisting of mostly social order and seek to study patterns of ordered events or behaviours. However, a functionalist believes that the best way to study the world is via objective approach while an interpretivist prefers subjective approach to study the world. On the other hand, radical structuralism and radical humanism are employed when researchers view the world as consisting of radical change and seek to understand or enact change. Radical structuralists prefer objective approach to study the world while radical humanists utilize subjective approach to perform their studies.

It is hereby argued that a single paradigm does not suffice in scientific social researches. This is because most of social researches deals with complex

phenomena and hence requires multiple paradigms to fully understand a given social phenomenon. Hence the need for multiple paradigms to study a given phenomenon in order to have an holistic and complete understanding of the phenomenon which can be achieved via the multi-paradigmatic approach to research. Ardalan (2010:34) agrees that insights generated by a single paradigm are partial and incomplete and hence the need to utilize multiple paradigms in order to complement a given study. Similarly, Bhattacharjee (2012:18) agrees that following complex nature of social phenomenon there are possibilities that a single research paradigm is partially correct and incomplete. This is in agreement with Ardalan (2010:35-36) who posits that researchers can gain much by exploiting new perspectives coming from other paradigms. Therefore the application of multiple paradigms leads to a better understanding of the multi-faceted nature of a given phenomenon.

Gravetter and Forzano (2009:147) state that the primary distinction between quantitative and qualitative research is the type of data they produce. Quantitative research produces numerical scores, such as durations, counts of incidents, ratings and scales. These are objective data. On the other hand, qualitative research is concerned with collection and analysing information in non-numeric form to produce narrative reports or written discussions of the observation. These are subjective data. Best and Khan (1989:89-90) agrees that qualitative and quantitative methods are not mutually exclusive and both can be mixed and used in one study of a given phenomenon. Moreover, Jick (1979:602) agrees that researchers can improve the accuracy of their judgments by collecting different kinds of data bearing on the same phenomenon, which leads to a concept known as triangulation. Olsen (2004) defines triangulation as the mixing of methods or data in order to allow diverse viewpoints to cast light upon a given topic.

Based on the fact that triangulation improves accuracy of judgments by considering diverse viewpoints from multi-paradigms, this study follows the triangulation approach. The theoretical framework of this research is underpinned and informed by the ontology and epistemology of both functionalism and interpretivism paradigms. The researcher's construct of reality and knowledge is not confined by one single paradigm but multi-paradigms. There are both strengths and weaknesses in each research paradigm (Figure 3.1). Hence, this study capitalizes on the strengths of a multi-paradigmatic approach where research paradigms complement one another in order to minimize the effects of weaknesses of a single paradigm.

3.3 Research methodology

Research methodology is about the logic behind methods used during research and techniques used in carrying that research (Welman *et al.*, 2005:2). According to Babbie and Mouton (2001), research methodology refers to a systematic, methodical and accurate execution of a research plan. Research methodology explains methods and techniques that a researcher use to attain knowledge about a given research problem in order to provide or suggest appropriate solutions regarding the research problem. Research methodology therefore explains how a researcher conducted a research.

Welman *et al.* (2005:6) explains that there are two approaches to research; the positivist approach and the anti-positivist approach. Welman *et al.* (2005:6) elaborates that the positivist approach is based on a philosophical assumption known as logical positivism where natural scientific methods are applied. The positivist approach suggests that a research must be limited to what is observed and measured objectively. Hence, what is observed and measured must exist independent of the feelings and opinions of individuals. According to Charmaraz (2006:188), the positivism approach subscribes to scientific methods consisting of objective systematic observation and experimentation in an external world. On the other hand, the anti-positivist approach argues if natural-scientific methods are the norms in social science studies. Anti-positivists argue that it is inappropriate to follow strict natural-scientific methods when collecting and interpreting data (Welman *et al.*, 2005:6). The anti-positivists approach focuses on the experiences of human behaviour.

Welman *et al.* (2005:6-9) further clarifies that research methodologies can be categorized into two major categories; namely qualitative and quantitative research. Qualitative research involves research methods where qualitative data such as descriptions of social life as observed by a researcher, unstructured interviews and data obtained from written sources are employed while quantitative research deals with quantitative data; which are numerical and hence measured and expressed in form of numbers (Jones, 2004a). Welman *et al.* (2005:8) mention that the purpose of quantitative research is to evaluate objective data consisting of numbers whilst qualitative research deals with subjective data produced by the minds of respondents or interviewees, mostly represented in language instead of numbers. Neuman (1997) mentions that qualitative data can be expressed as words, objects and

pictures while Babbie and Mouton (2001) states that quantitative data involves variables that are used to measure against the research problem. Therefore this study utilizes both qualitative and quantitative epistemological methodologies using interviews and survey respectively. As mentioned above, this is triangulation.

The following section elaborates on techniques used in requirement analysis as an attempt to indicate typical research techniques that could be incorporated in the research design of this study.

3.4 Typical research techniques used in requirement analysis

According to Sommerville (2007:153), an effective interviewer should be open-minded and a good listener who is willing to listen to stakeholders in order to understand user requirements. Furthermore, according to Sommerville (2007:153) an interviewer should be able to prompt and lead interviewees into giving specific information as opposed to leaving interviewees with general information. Sommerville (2007:152-158) and Jiang *et al.* (2008:326) further describe about interviews, use cases, prototype and ethnography as techniques used to facilitate requirements analysis. Olson (2004:93) reports about meetings, interviews and brainstorming as some of techniques used to elicit user requirements. These techniques can be stated as follows:

- i. Interviews refers to a two way instant communication or meeting where one party called interviewer poses questions to another party called interviewee, who answers the asked questions. Interviewers can for instance be researchers while interviewees can be software developers and stakeholders such as end users and managers.
- ii. Use cases, according to Bennett *et al* (2006: 145), refers to descriptions of functionality of the system from the users' perspective. Use cases identify users or actors and the type of interactions between users and the system. Use cases are mostly used in a modelling language called UML (Unified Modelling Language) to identify actors (users) and to show how actors interact with the software system.
- iii. Prototype refers to a partially developed system whose purpose is to provide users with means to identify user requirements. According to Hughes and

Cotterell (2006:78-80) prototype is ideal for discovery of user requirements where the degree of uncertainty is high. Users can test, explore and understand different ideas using prototypes.

- iv. Ethnography is a technique where an individual observe employees at their work (Hughes & Cotterell, 2006:453). For instance, a software developer could visit employees and observe their behaviour at work and how they actually work in order to identify user needs that can be satisfied by a software system.
- v. Brainstorming refers to a process where individuals generate ideas with the intent to seek to understand the situation at hand (Olson, 2004:98). The purpose is to understand the situation in order to implement appropriate solutions to identified problems. According to Olson (2004:98), brainstorming allows the generation of better ideas than individual thoughts. When done correctly in a climate of free association, brainstorming taps into the mind of more people, which results into better ideas. Hence, brainstorming is the idea generation process.
- vi. Task and user analysis involves and focuses on deep understanding of how users perform their tasks (Stary, 2002:437). This understanding includes users' goals, personal, social and cultural characteristics, influence of environment, previous experience and knowledge as well as what users value as being important to make execution of tasks a delightful experience.

Studies on competencies have a lot in common in terms of methodology and objectives. Ahn and McClean (2008:545) used interviews to identify competencies required for port and logistics personnel in Busan, South Korea. Thereafter, Ahn and McClean (2008:547) had to use survey questionnaires to cover large samples in order to validate their research on larger population.

According to Le Deist and Winterton (2005:31), competency framework is a result of observing successful and effective job performers and determining what makes these individuals to differ from less successful performers. Frank (2005) reports about intensive steps involved to identify competencies of physicians and medical doctors. Frank (2005) mentions that methodologies applied to identify and determine competencies of physicians and doctors include consultation among medical

specialties such as physicians and surgeons as well as among doctors and patients. These are basically meetings and interviews among physicians and the society. The research was however complemented by surveys to test the validity of the competency framework (Frank, 2005).

As mentioned in chapter one, the study on master's students of computer science who were assigned software development tasks utilized ethnography as the methodology (Rivera-Ibarra, Rodriquez-Jacobo, Fernandez-zepeda & Serrano-Vargas, 2010:35). According to Turley and Bieman (1995:3), during phase one of their research on competencies of software engineers, they used interviews to identify essential competencies of software engineers in a software development company in United States of America (USA). Interviews with software engineers were recorded and transcripts were made. Thereafter Turley and Bieman (1995) applied content analysis to analyse information collected from software engineers. During phase two, Turley and Bieman (1995:11) had to use survey questionnaires to cover larger population of software engineers; also known as software developers.

The following section describes research design for this study.

3.5 Research design for this study

Based on the above, it is clear that social science studies tend to use both qualitative and quantitative methodologies together; the process known as triangulation (De Vos *et al.*, 2002:341-342). Yin (2003) defines triangulation as a process where both qualitative and quantitative methods are used to facilitate the research process for a given research study. Olsen (2004) defines triangulation as the mixing of data or methods in order to allow diverse viewpoints to cast light upon a given topic. Olsen (2004) further sub-divides triangulation into two categories; data triangulation (mixing data types) and methodological triangulation (mixing methodologies).

The effectiveness of triangulation is based on the premise that weaknesses in one method are compensated by strengths of another method (Jick, 1979:604). Jick (1979:602) explains that qualitative and quantitative methods are complementary and not rival methods. Furthermore, Olsen (2004) reports that mixing methodologies such as the use of survey and interviews is a more profound form of triangulation. Internet-based surveys are popular because they are economical in terms of time and costs. In addition, Internet-based surveys facilitate access to large research populations

and samples (Burns & Burns: 2008:495). On the other hand, interviews help in obtaining in depth and specific understanding of a given phenomenon (Welman *et al.*, 2005:166).

As such, this research employs the triangulation approach using survey questionnaires and interviews to accomplish its objectives. Both survey questionnaires and interviews aim at obtaining information pertaining to tasks performed by software developers, tools used and skills required to become a competent software developer, as indicated in figure 3.2 in the following section.

As illustrated in the section below, this study is accomplished via both quantitative and qualitative methodologies via survey questionnaires and interviews.

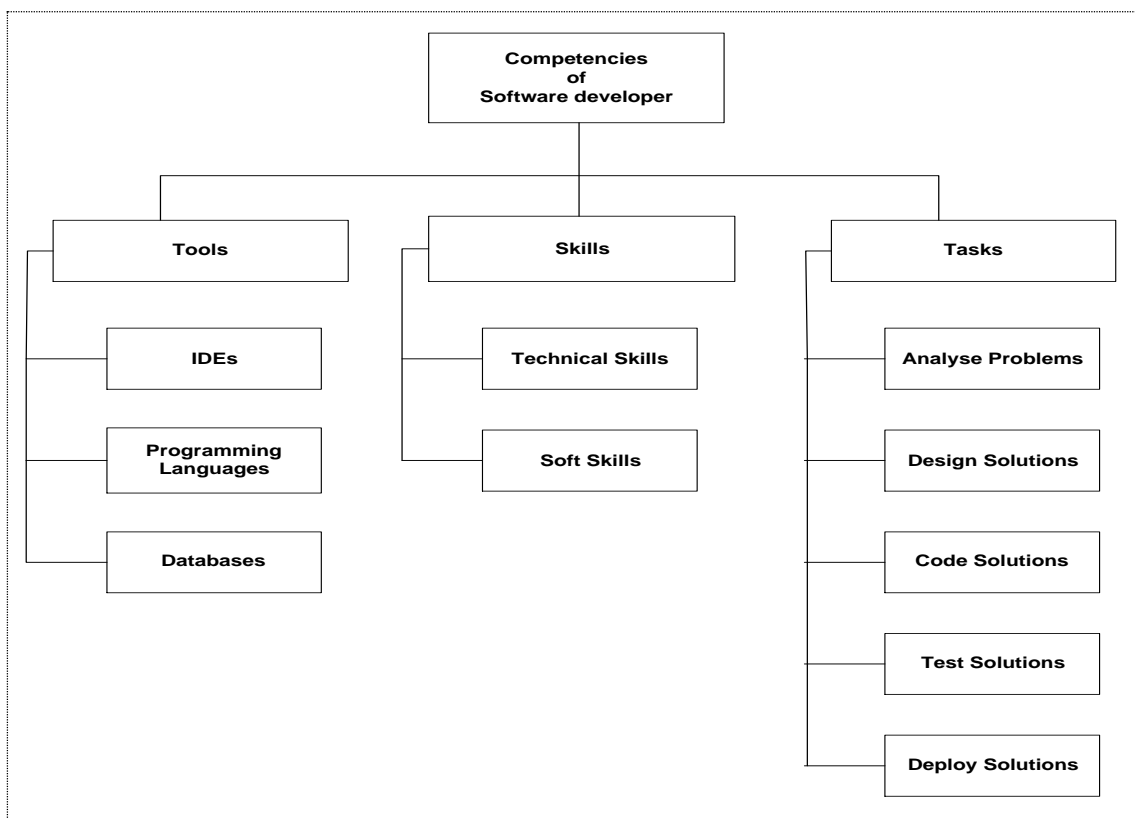


Figure 3.23: Competencies of software developers under investigation

3.5.1 Survey questionnaire

The purpose of this research is to provide answers to the research question “What are fundamental competencies of a software developer?” In order to answer this question, the question was divided into three sub-questions as follows:

- i. How is software developed?*
- ii. What technologies and tools are used to develop software?*

iii. What are essential skills required in order to develop software?

Based on the research question, the researcher designed a questionnaire composed of twenty one questions; twenty closed-ended questions and one open-ended question. Closed-ended questions search for answers regarding all necessary statistical data required to understand the research population such as gender, years of experience in software development, level of skills and programming languages mostly used. Moreover, closed-ended questions seek to find answers about the software development process, technologies and tools used to develop software, tasks performed by software developers and important skills required by software developers. Questions on skills were also addressed by another open-ended question. This question was set in such a manner as to give freedom to research participants to provide as much rich information as possible without being limited.

A survey questionnaire was uploaded on a website managed by Survey Gizmo; a well-known online survey services provider. Then a link to the research survey was distributed among software developers via emails. The Silicon Cape, a Western Cape based online community whose main responsibility is to promote Information Technology in South Africa, collaborated with the researcher and participated in reaching out and distributing the survey link among software developers.

The CITI (Cape Information Technology Initiative) an organization whose main mandate is to promote Information Technology in the Western Cape Province collaborated with the researcher and participated in distributing the link to survey among software companies and business organizations within the Western Cape Province.

The co-operation office in the department of Information Technology at Cape Peninsula University of Technology collaborated with the researcher and participated in distributing the survey link to software companies and business organizations running internship programmes via the Cape Peninsula University of Technology. The co-operation office has long history of assisting students with internship programmes and permanent employment as software developers. Hence, the researcher utilized the good work relationship between the co-operation office and the industry in order to reach more software developers in the industry.

Some recruitment agencies whose main function is to find employment for software developers were contacted and involved in distributing the survey link among companies employing software developers.

Part time students from the Cape Peninsula University of Technology who are working as software developers participated in the research. In collaboration with the researcher, these software developers also distributed the survey link among other software developers in their respective work places.

The Chief Information Officer (CIO) forum, post-graduate initiative headed by the Cape Peninsula of Technology and comprised of members occupying senior positions in business organization and software companies in the Western Cape, participated in promoting the research and distributing the survey link among software developers.

Identifiable sponsors, well-designed instruments, financial incentives and repeated contacts are among the steps that can be used to improve results of data collection from Internet surveys (Fowler, 2009:61). Using identifiable contact people from the industry, the researcher performed several follow-ups and sent several reminders via telephone and emails to the research population. As indicated on Appendix E, the researcher contacted several software companies and business organizations to promote awareness of the research and requesting participation of software developers to the research, the initiative that was welcomed by most of business organizations and software companies. Moreover, the researcher utilised Twitter and Facebook, well-known social sites, to target software developers on the online community.

Ultimately, a link to access the survey questionnaire was distributed and made available to software developers. The survey was made accessible for the duration of six months; from 07 September 2010 to 07 March 2011. Data were collected and analysed using PASW⁸ Statistics 18 and Excel 2010.

⁸ PASW: software package used for quantitative data analysis

3.5.2 Interviews

In order to complement the study, unstructured interviews were performed. Twelve software developers were interviewed during the months of March, April and May of 2012. The interviews were performed on one to one basis where a researcher interviewed one individual after another. All interviewees were software developers with working experience from five years, working as senior software developers. Each interview session took an average of one hour depending on the availability of an interviewee. The interviews were unstructured and hence interviewees were open to discuss and share anything related to competencies of software developers. However, following time limitations the scope of interviews was based on the following three questions:

- i. As a software developer, what kind of software projects are you involved in?*
- ii. As a software developer, what technology do you use in your company?*
- iii. As a software developer, what skills do you have in order to perform your tasks in your software projects?*

The above questions aim at identifying and understanding competencies of software developers in the Western Cape Province.

Six companies participated in the interviews and two interviewees were taken from each company. Companies participated are four software companies and two business companies. In this study, a software company is any company whose major business activities are to develop software systems as commercial products. These companies make profit primarily by selling software products and by providing software consulting and supporting services. Such software companies are characterized by having varieties of software projects and normally tend to have different project teams undertaking different software projects in a given time. Referring to companies interviewed the number of software developers in a project team ranged from five to fifteen.

On the other hand, business companies that participated in this research are those companies that use software solutions to execute their business activities. These companies do not sell software products as a business rather they utilize software technology to execute most of their business activities. Software projects for these companies are mostly small and managed by small project teams. Most of tasks are

performed individually and there are no structured project teams. In this case, software developers normally focus on maintenance and addition of new small features to existing software systems. Often whenever business companies have business requirements that require a big software project, they tend to utilize services and expertise offered by consulting software companies.

Data collected during interviews was analysed using the content analysis technique. The researcher had to read text of data collected to identify patterns and themes. Content analysis as a data analysis technique is described in details in chapter four.

3.6 Research population and sampling

Mouton (1996:134) refers to population as a collection of objects, events or individuals sharing common characteristics or behaviour that need to be studied by a researcher. According to Welman *et al.* (2005:53), and Babbie and Mouton (2001:100) a population is a group of potential participants to whom a researcher wants to generalise the results of a study or wishes to make specific conclusions. Welman *et al.* (2005:55) also mention that it is impractical and uneconomical to include all members of the population in a research project. Hence, the need for population sampling where a representative portion of the population is selected and used for study to generalize conclusions on the research population.

Sampling refers to a process of selecting a particular set of elements from a population to be studied in order to achieve research aims and draw conclusions about the whole population (Babbie & Mouton, 2001; Kumar, 2005). Welman *et al.* (2005:53) mention that a sample selected to represent a particular research population is also known as a “unit of analysis”. According to Welman *et al.* (2005:56), there are two categories of sampling; probability and non-probability sampling. With probability sampling, it is possible to ascertain that any member of the population have equal chance of being included in the study. This is possible if the number of the research population is specifically known. On the other hand, non-probability sampling is based on the selection of unknown or not readily identifiable research participants for a given population (Singleton & Straits, 2005).

The population under this research is all software developers in the Western Cape Province. Since the number of software developers in the whole Western Cape

Province is not known and cannot be easily quantified, non-probability sampling was employed during this research. Specifically, convenience sampling, which is a non-probability sampling that deals with selecting readily available research subjects, according to Welman *et al.* (2005:69), was used in this research. The objective is to learn from experienced software developers on what are competencies of software developers. The survey questionnaire that was distributed among software developers was structured in such a way that it was possible to establish if a research participant was experienced in terms of executing software development tasks. All levels of software developers namely; junior, middle and senior level, were issued questionnaires indiscriminately. While inputs were gathered from all software developers, answers pertaining to skills of software developers were primarily obtained from experienced software developers. Turley and Bieman (1995:121) report that software engineers who exhibited exceptional performance had extensive work experience while non-exceptional software engineer were said to be inexperienced. Therefore, there is a correlation between skills and work experience.

3.7 Reliability and validity of research methodology

Findings of a given research on a specific unit of analysis ultimately need to be applied to the entire research population. Hence, the need of reliability and validity of research methods and instruments used to perform a given research. A research instrument is anything, for instance a questionnaire that a researcher can use in order to undertake a research (Hofstee, 2006:115). Sarantakos (1998) refers to reliability as how dependable or consistent are the instruments used during research. According to Welman *et al.* (2005:145) reliability is the extent to which scores obtained during a particular measurement may be generalised to different measuring occasions, measurements/tests forms and measurement/test administrators. For reliable measurements, scores that are assigned to individuals should therefore be consistent irrespective of the time of measurements, the test used, and the person administering the test (Welman *et al.*, 2005:145). However, reliability is more applicable in experimental research where taking measurements is a common procedure during data collection.

Validity is the extent to which the research findings accurately represent what is really happening in the research population (Welman *et al.*, 2005:142). Furthermore Welman *et al.* (2005:106-131) report on internal validity and external validity. Internal validity has something to do with whether changes in a dependent variable are due to

changes on independent variable and not something else. Welman *et al.* (2005:125-128) further divide external validity into ecological validity and population validity. Ecological validity refers to the generalisation to a relevant universe of conditions and is more critical in experimental researches. On the other hand, population validity refers to the degree to which the findings obtained for a sample may be generalised to the total population to which the research hypothesis applies (Welman *et al.*, 2005:125). Hence, population validity is of critical importance in this research.

Prior to constructing a questionnaire for this research, the researcher performed a considerable research on software development and on previous competency studies. While competency studies are different, their main objective is to determine competencies in a given field of speciality. For instance, studies on competencies of software engineers in the USA, competencies of medical professions in Canada, competencies for supervisors in the building and construction sector in Chile and competencies for employees for American Medical System company (Turley & Bieman, 1995; Frank, 2005; Serpell & Ferrada, 2007; Gangani *et al.*, 2004). This is to ensure that this study conforms to standards of competency studies performed by other researchers. This is in order to improve population validity such that research findings for this study can be generalised to all software developers in the Western Cape Province.

Questions in the survey questionnaire were geared to find answers to research sub-questions as specified on section 3.4.1 above. Moreover the questionnaire was based on previous researches that have been validated. However, the researcher performed adjustments in order to ensure that questions depicted the context of the research population in the Western Cape Province. The questionnaire was distributed among software developers and hence answers to survey were provided by software developers developing software systems in the Western Cape Province.

In order to reinforce and ensure validity and reliability of the questionnaire, the first draft of questionnaire was issued to ten randomly selected software developers from five randomly selected business companies. This was to ensure that the content of questionnaire is understood among software developers and that it measures what is expected to be measuring. The selected ten software developers were contacted for their feedback with regards to the content of questions on the survey. All feedback were investigated by the researcher and incorporated into the research to prepare a

final questionnaire. Lastly the final questionnaire was uploaded on the website managed by Survey Gizmo, well-known online survey service providers. Ultimately the survey link was made accessible among software developers in the Western Cape Province.

3.8 Ethics, Consent and Confidentiality

Before the survey was distributed and interviews performed, approval from the Post Graduates Ethics Committee of the Cape Peninsula University of Technology was obtained. All research participants were informed of their rights as far as this research is concerned. This included answering all questions or only questions where the research participant sees fit and the right to unconditional withdrawal from the research. Since participation in this research was voluntary, research participants had rights to decide to participate or not to participate. Furthermore, it is acknowledged that information provided by research participants is for the sole purpose to enable the researcher to complete the fulfilments required to obtain the master's degree in information technology. Data collected will not be provided to the public or any external party for other purposes. All information including identifying information such as name, telephone and email addresses will be kept confidential and research participants will remain anonymous.

3.9 Chapter Summary

This chapter introduced research paradigms, explained on categories of research methodologies and briefly discussed about typical research design followed by previous studies on competencies. The research design followed during this research was discussed. Population and sampling techniques were elaborated. Furthermore, ethics, consent and confidentiality matters were discussed.

3.10 Conclusion

This research took advantage of triangulation using survey questionnaire to measure quantitative data and interviews to obtain qualitative data. Interviews were performed among selected software developers. In addition, survey questionnaire was uploaded on a website and a website link to access the survey was distributed among software developers. Both quantitative and qualitative methodologies were employed.

The following chapter four discusses data analysis.

CHAPTER FOUR: DATA ANALYSIS AND PRESENTATION

“It is a capital mistake to theorize before one has data”

(Arthur Conan Doyle)

4.1 Introduction

This chapter presents data collected during the research. Data were collected using survey questionnaires and interviews. Quantitative data collected via questionnaires were analysed using Excel 2010 and PASW software while qualitative data of open-ended questions and interviews were analysed using the content analysis technique.

The chapter presents quantitative data analysis in section 4.2 and qualitative data analysis in section 4.3. The chapter concludes with chapter summary in section 4.4 and conclusion in section 4.5.

4.2 Quantitative data analysis

With reference to Table 4.1 and Figure 4.1 below, a total of 263 survey responses were collected from software companies and business organizations. Fifty-seven (57) responses were incomplete because respondents did not answer all compulsory questions or they were not software developers from the region under study. For instance, there were a few responses from Asia and Europe. Those incomplete responses were excluded from this study. As a result a total of two hundred and six questionnaire responses (n=206) were eligible for data analysis; these were complete questionnaires because respondents answered all compulsory questions and belong to the population under study.

Table 4.1: Response status of questionnaire (n=263)

	Number of respondents	Percentage
Complete	206	78.3%
Incomplete	57	21.7%
Total	263	100.0%

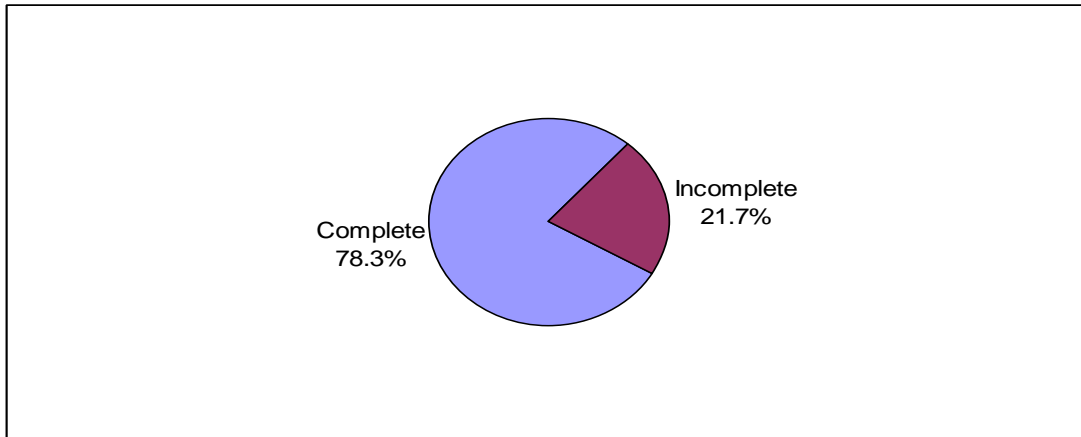


Figure 4.1: Response status of questionnaire (n=263)

Data analysis was performed using Microsoft Office Excel 2010 and PASW Statistics 18. Analysed data is represented by using tables and figures. A table indicates analysed data in the form of numbers while a figure translate collected data to give a generic pictorial view of data. While a table and a figure may achieve one purpose, they tend to give different views regarding same data. Therefore, in this study, data for each question are presented by both a table and a figure. A table is comprised of three columns where the first column represents a variable to be measured. A variable is a characteristic or an attribute of the study object (Welman *et al.*, 2005:16). The second column represents frequency or number of counts for the variable. The third column uses percentage to indicate to what extent does the frequency in second column form part of the whole community of 206 software developers who participated in this research. Regarding figures, depending on the question, either a pie chart or bar chart is used to present the generic pictorial view of data analysis.

4.2.1 Profile of software developer

This section presents demographical data collected in order to understand the profile of software developers who participated in this study. Data collected are related to gender, age, race, citizenship, work experience, education background and respondent's technical level in software development. This data is mutually exclusive and hence a respondent could provide only one answer per question.

4.2.1.1 Gender of respondents

Question: What is your gender?

With reference to Table 4.2 and Figure 4.2, the majority 80.6 percent of respondents are male and only 19.4 percent of respondents are female.

Table 4.2: Gender (n=206)

	Number of respondents	Percentage
Male	166	80.6%
Female	40	19.4%
Total	206	100.0%

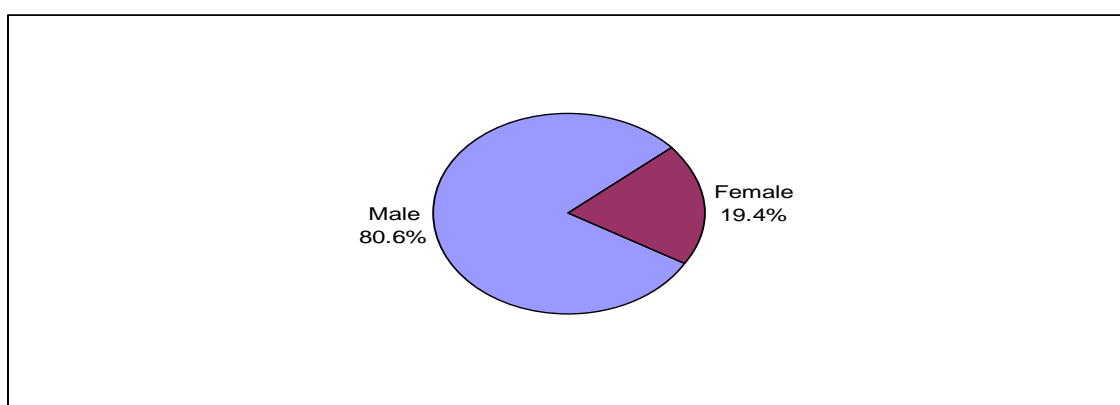


Figure 4.2: Gender of respondents (n=206)

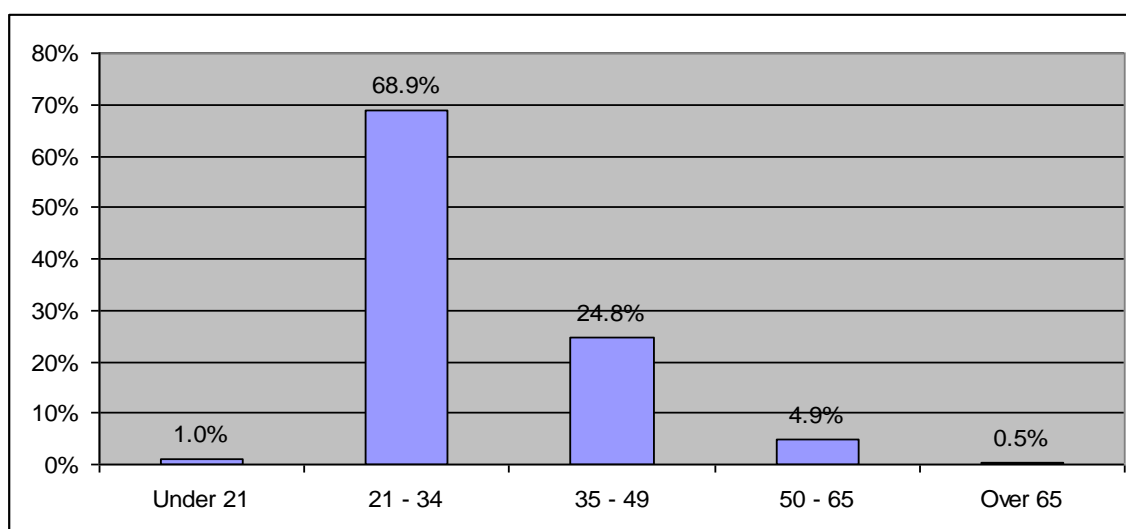
4.2.1.2 Age of respondents

Question: What is your age?

With reference to Table 4.3 and Figure 4.3 in the following section, the majority 68.9 percent of respondents are between 21 years to 34 years. The second major age group is 35 years to 49 years, which constitutes 24.8 percent of respondents. 4.9 percent of respondents are between 50 years to 65 years. One percent of respondents are under 21 years and only 0.5 percent of respondents are over 65 years.

Table 4.3: Age in years (n=206)

	Number of respondents	Percentage
Under 21	2	1.0%
21 – 34	142	68.9%
35 – 49	51	24.8%
50 – 65	10	4.9%
Over 65	1	0.5%
Total	206	100.0%

**Figure 4.3: Age in years (n=206)**

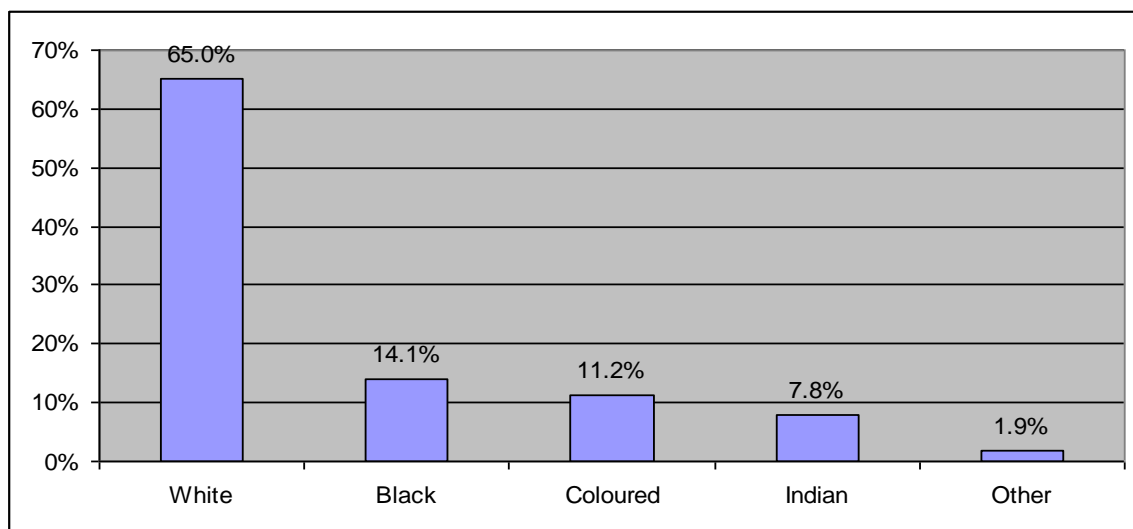
4.2.1.3 Race of respondents

Question: What is your race?

South Africa population is divided into four major races; black, white, coloured and Indian (Bornman, 2006:387). With reference to Table 4.4 and Figure 4.4 in the following section, the majority 65 percent of respondents are white followed by black at 14.1 percent and coloured at 11.2 percent. Indian constitutes 7.8 percent and only 1.9 percent of respondents classify themselves as other races rather than the above-mentioned four races.

Table 4.4: Race of software developer respondents (n=206)

	Number of respondents	Percentage
White	134	65.0%
Black	29	14.1%
Coloured	23	11.2%
Indian	16	7.8%
Other	4	1.9%
Total	206	100.0%

**Figure 4.4: Race of respondents (n=206)**

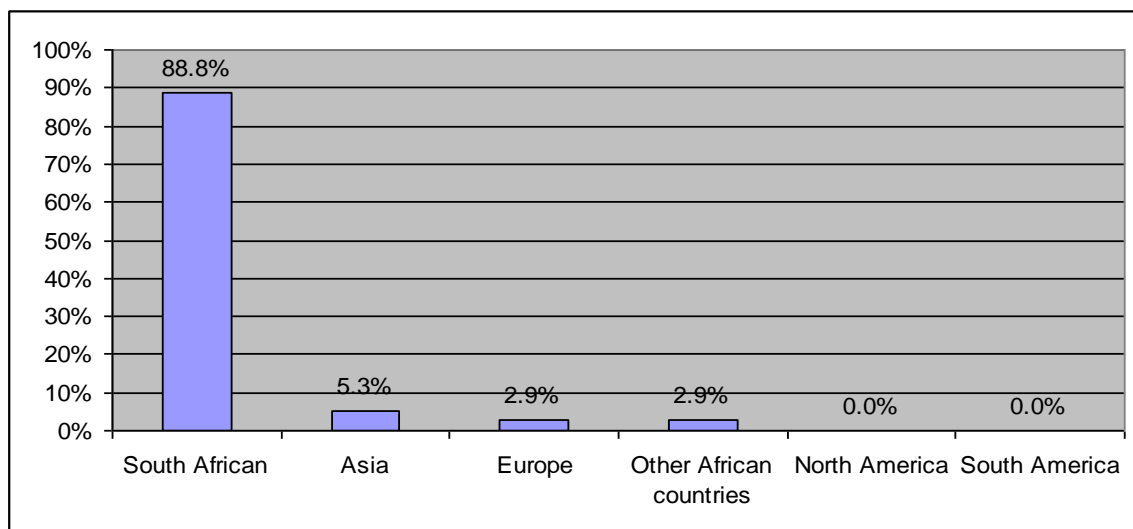
4.2.1.4 Citizenship of software developer respondents

Question: What is your citizenship?

With reference to Table 4.5 and Figure 4.5 in the following section, the majority 88.8 percent of respondents are South Africans followed by 5.3 percent as Asian people. Europe and other Africa countries have a tie of 2.9 percent each. According to data collected, there are neither North Americans nor South Americans who participated in this research.

Table 4.5: Citizenship of respondents (n=206)

	Number of respondents	Percentage
South Africa	183	88.8%
Asia	11	5.3%
Europe	6	2.9%
Other African countries	6	2.9%
North America	0	0.0%
South America	0	0.0%
Total	206	100.0%

**Figure 4.5: Citizenship of respondents (n=206)**

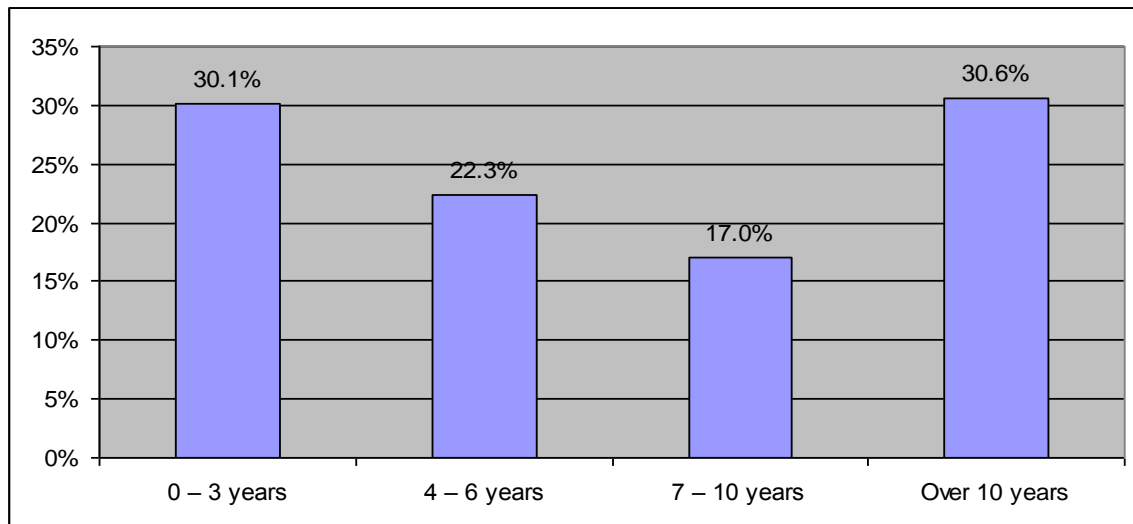
4.2.1.5 Work experience of software developer respondents

Question: What is your work experience as a software developer?

With reference to Table 4.6 and Figure 4.6 in the following section, the majority 30.6 percent of respondents have been working as software developers for over 10 years. Thirty (30.1) percent report to have worked for not more than 3 years. Twenty-two (22.3) percent have work experience of between 4 years to 6 years. The least group constitutes 17 percent of respondents who report to have work experience of between 7 years to 10 years.

Table 4.6: Work experience in years (n=206)

	Number of respondents	Percentage
0 – 3 years	62	30.1%
4 – 6 years	46	22.3%
7 – 10 years	35	17.0%
Over 10 years	63	30.6%
Total	206	100.0%

**Figure 4.6: Years of work experience (n=206)**

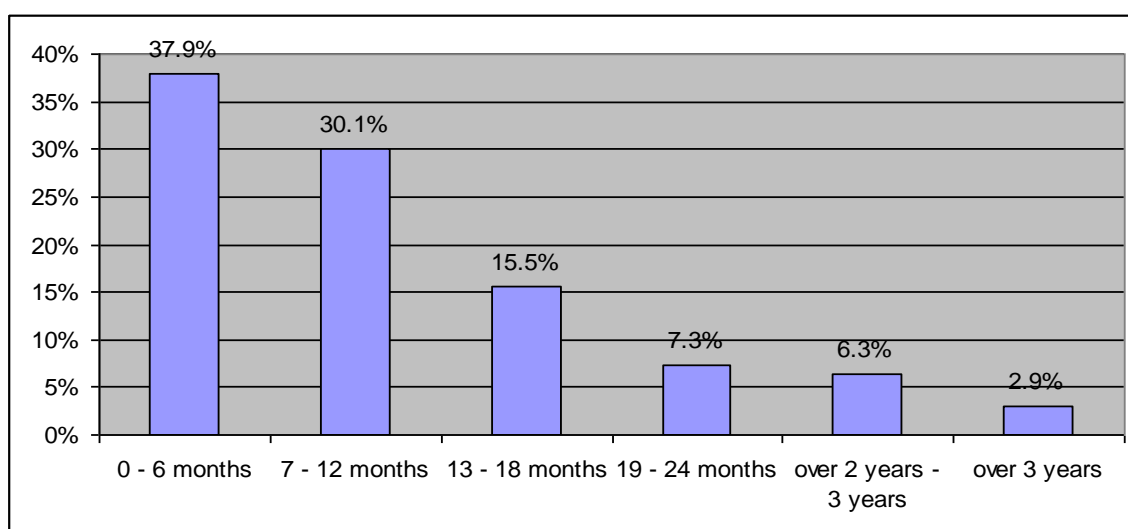
4.2.1.6 Time taken to become self-dependent software developer

Question: How long did it take you to become a software developer who performs tasks with minimal assistance from other software developers?

With reference to Table 4.7 and Figure 4.7 in the following section, the majority 37.9 percent of respondents indicate that it took them a maximum of 6 months to become proficient with software development tasks. The second major group constitutes 30.1 percent of respondents and reports to have taken between 7 months to 12 months to become proficient software developers. 15.5 percent of respondents indicate to have taken between 13 months to 18 months. Seven (7.3) percent indicate to have taken 19 months to 24 months. 6.3 percent took between 2 years to 3 years while 2.9 percent took over 3 years.

Table 4.7: Time taken to become self-dependent (n=206)

	Number of respondents	Percentage
0 - 6 months	78	37.9%
7 - 12 months	62	30.1%
13 - 18 months	32	15.5%
19 - 24 months	15	7.3%
over 2 years - 3 years	13	6.3%
over 3 years	6	2.9%
Total	206	100.0%

**Figure 4.7: Time taken to become self-dependent (n=206)**

4.2.1.7 Software development certification status

Question: Are you a certified software developer, certified with software training authorities like Microsoft, Oracle or Sun?

With reference to Table 4.8 and Figure 4.8, the majority 71.4 percent of respondents are not certified software developers while only 28.6 percent of respondents report to be certified software developers.

Table 4.8: Time taken to become self-dependent (n=206)

	Number of respondents	Percentage
Yes	59	28.6%
No	147	71.4%
Total	206	100.0%

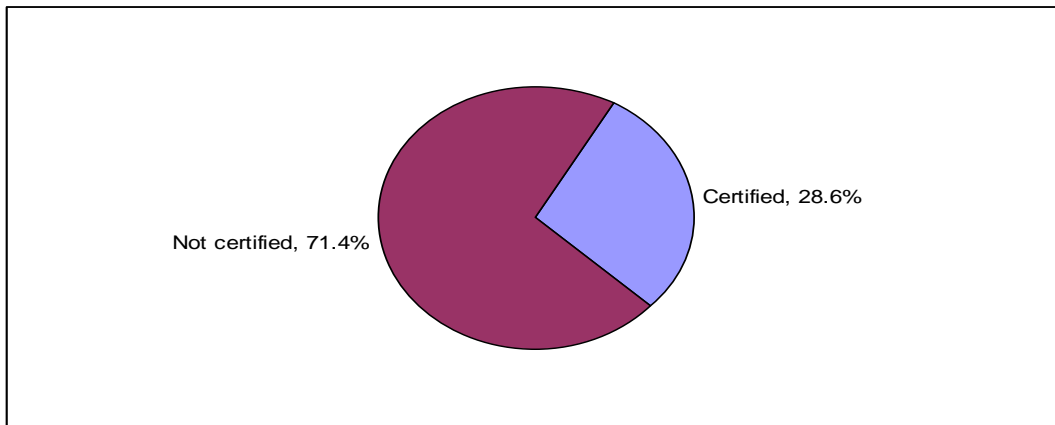


Figure 4.8: Software development certification status (n=206)

4.2.1.8 Education background of respondents

Question: What is your highest education qualification?

With reference to Table 4.9 and Figure 4.9, the majority 37.9 percent of respondents have a bachelor degree. Thirty-five (35) percent have diploma followed by 13.1 percent who have certificates. 7.3 percent of respondents report to have master's degree while only 1 percent indicates to have PHD. However, 5.8 percent of respondents indicate to have been software developer without formal education.

Table 4.9: Education background of respondents (n=206)

	Number of respondents	Percentage
Certificate	27	13.1%
Diploma	72	35.0%
Bachelor degree	78	37.9%
Master's degree	15	7.3%
PHD	2	1.0%
Other	12	5.8%
Total	206	100.0%

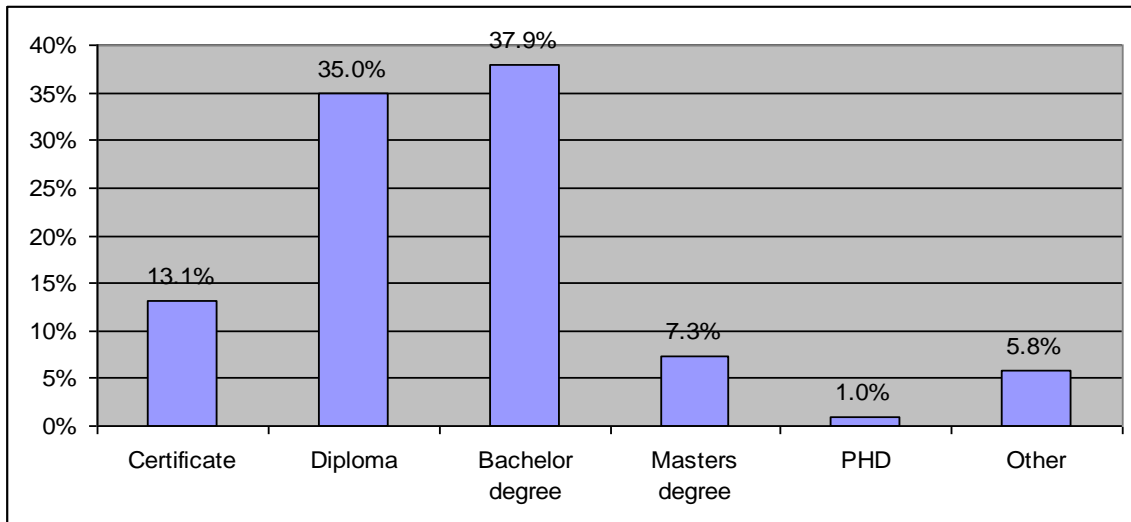


Figure 4.9: Education background of respondents (n=206)

4.2.1.9 Current position of respondents

Question: What is your current technical level (position) as a software developer?

With reference to Table 4.10 and Figure 4.10, the majority 52.9 percent of respondents indicate to be senior software developers. 25.7 percent have reached middle level position while 21.4 percent report to be junior software developers.

Table 4.10: Current level as a software developer (n=206)

	Number of respondents	Percentage
Junior	44	21.4%
Middle	53	25.7%
Senior	109	52.9%
Total	206	100.0%

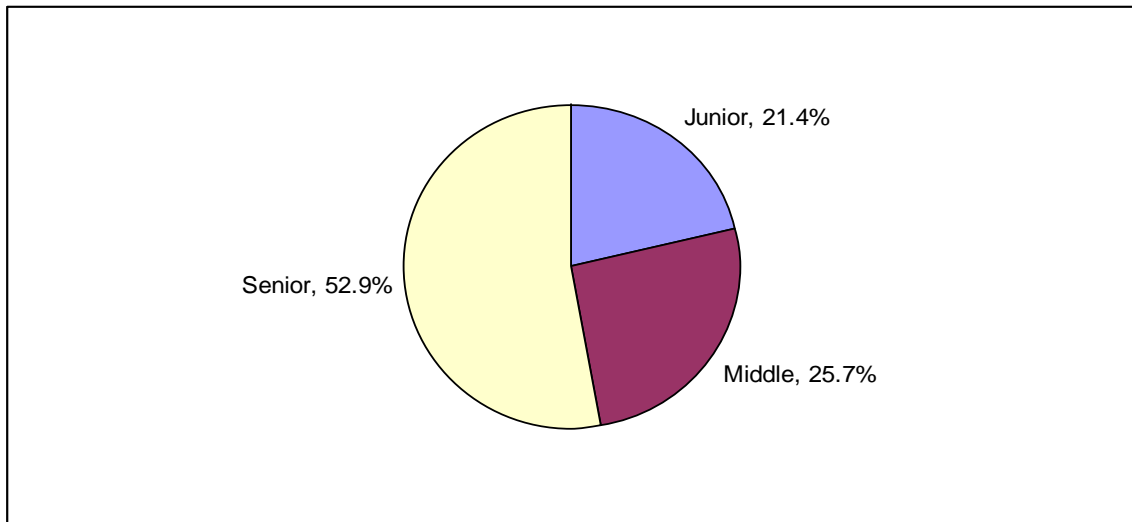


Figure 4.10: Current position of software developer (n=206)

4.2.1.10 Province of respondents

Question: In which province are you coming from?

With reference to Table 4.11 and Figure 4.11, the majority 80.6 percent of respondents are coming from the Western Cape Province. This was the unit of analysis for this research. However, 14.1 percent of respondents indicate that they are coming from Gauteng while 2.4 percent of respondents are from KwaZulu Natal. Eastern Cape and Limpopo have a tie of 1 percent each. Moreover, Free State and North West have a tie of 0.5 percent each.

Table 4.11: Provinces of respondents (n=206)

	Number of respondents	Percentage
Western Cape	166	80.6%
Gauteng	29	14.1%
KwaZulu Natal	5	2.4%
Eastern Cape	2	1.0%
Limpopo	2	1.0%
Free state	1	0.5%
North West	1	0.5%
Total	206	100.0%

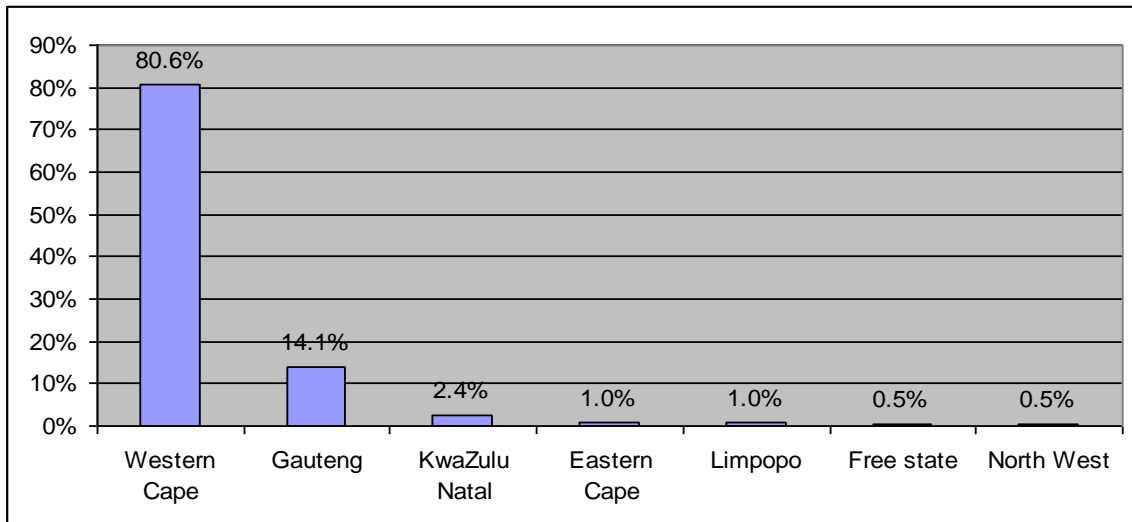


Figure 4.11: Provinces of respondents (n=206)

4.2.2 Software development technologies

This section presents software development technologies used by respondents while performing their tasks as software developers. Six questions were used to investigate and address software development technologies. The questions covered integrated development environment (IDE) tools, programming languages, database management systems (DBMS), code version control tools, application servers and software deployment environments. Data regarding technologies is not mutually exclusive since a software developer may use more than one technology or tool to perform a particular task. Therefore, a research respondent may provide more than one answer per question.

4.2.2.1 Software development environment tools

Question: What among these Software Development Environment (IDE) tools, do you use most of the time?

Ten commonly used IDE tools were identified and respondents were asked to indicate whether these tools are being used while performing their tasks as software developers. With reference to Table 4.12 and Figure 4.12 in the following section, there are five IDE tools where each IDE tool is used by more than 10 percent of respondents.

The most used top five IDE tools are:

- i. *Microsoft visual studio used by 52.9 percent of respondents*
- ii. *Notepad used by 31.6 percent of respondents*
- iii. *Net beans used by 18 percent of respondents*
- iv. *Eclipse used by 16.5 percent of respondents*
- v. *Dream weaver used by 15 percent of respondents.*

However, 46.6 percent of respondents indicate to use other tools. This could be in-house tools, which are specific to companies and used in conjunction with other tools. Each of the following five tools: Microsoft SharePoint, ActiveState Komodo, Jcreator, Xcode and PhpED is used by less than 10 percent of respondents where PhpED is the least used IDE, used by 2.4 percent of respondents.

Table 4.12: Software development IDE (n=206)

IDE tool	Number of responses	Percentage
Microsoft visual studio	109	52.9%
Other	96	46.6%
Notepad	65	31.6%
Net beans	37	18.0%
Eclipse	34	16.5%
Dreamweaver	31	15.0%
Microsoft share point	20	9.7%
Activestate komodo	7	3.4%
Jcreator	6	2.9%
Xcode	5	2.4%
PhpED	5	2.4%

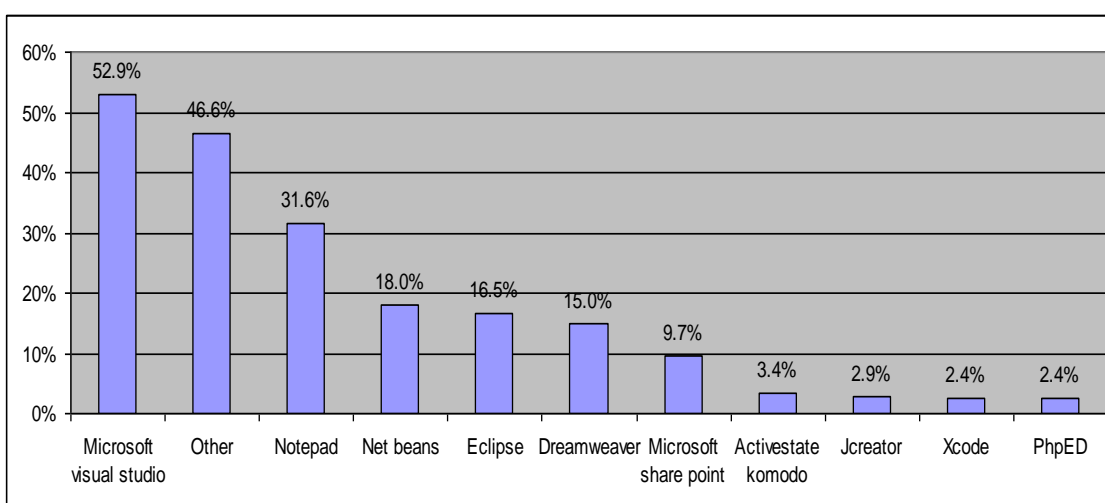


Figure 4.12: Software development tools (n=206)

4.2.2.2 Programming technologies

Question: What among these programming technologies, do you use most of the time?

Twenty-two mainstream programming technologies were identified. In this study, a programming technology refers to any software technology used to write computer instructions, in the form of codes, that when processed form part of software system. This includes programming languages and other client side and server side programming technologies. Client side refers to technologies used primarily for the design and functioning of user interfaces while server side technologies refer to programming of processes to be executed by the server machines. With reference to Table 4.13 and Figure 4.13 in the following section, there are twelve programming technologies where each technology is used by more than 10 percent of respondents. The most used top twelve programming technologies are:

- i. *HTML used by 55.8 percent of respondents,*
- ii. *Java script used by 46.6 percent of respondents,*
- iii. *CSS used by 46.1 percent of respondents,*
- iv. *XML used by 43.7 percent of respondents,*
- v. *C# used by 42.7 percent of respondents,*
- vi. *PHP used by 31.6 percent of respondents,*
- vii. *ASP.NET used by 29.1 percent of respondents,*
- viii. *Ajax used by 28.6 percent of respondents,*
- ix. *Java used by 20.4 percent of respondents,*
- x. *Visual basic used by 17 percent of respondents,*
- xi. *UML used by 11.7 percent of respondents*
- xii. *VB script used by 10.7 percent of respondents.*

However, 33.5 percent of respondents indicate to use other programming technologies. Each of the following ten programming languages: COBOL, Ruby, C++, Python, Perl, Delphi, Assembler, Pascal, C and FORTRAN is used by less than 10 percent of respondents where FORTRAN is the least used programming language, used by 0.5 percent of respondents.

Table 4.13: Programming technologies (n=206)

	Number of responses	Percentage
HTML	115	55.8%
Java script	96	46.6%
CSS	95	46.1%
XML	90	43.7%
C#	88	42.7%
Other	69	33.5%
PHP	65	31.6%
ASP.NET	60	29.1%
Ajax	59	28.6%
Java	42	20.4%
Visual basic	35	17.0%
UML	24	11.7%
VB script	22	10.7%
COBOL	18	8.7%
Ruby	13	6.3%
C++	11	5.3%
Python	10	4.9%
Perl	8	3.9%
Delphi	6	2.9%
Assembler	5	2.4%
Pascal	4	1.9%
C	3	1.5%
FORTTRAN	1	0.5%

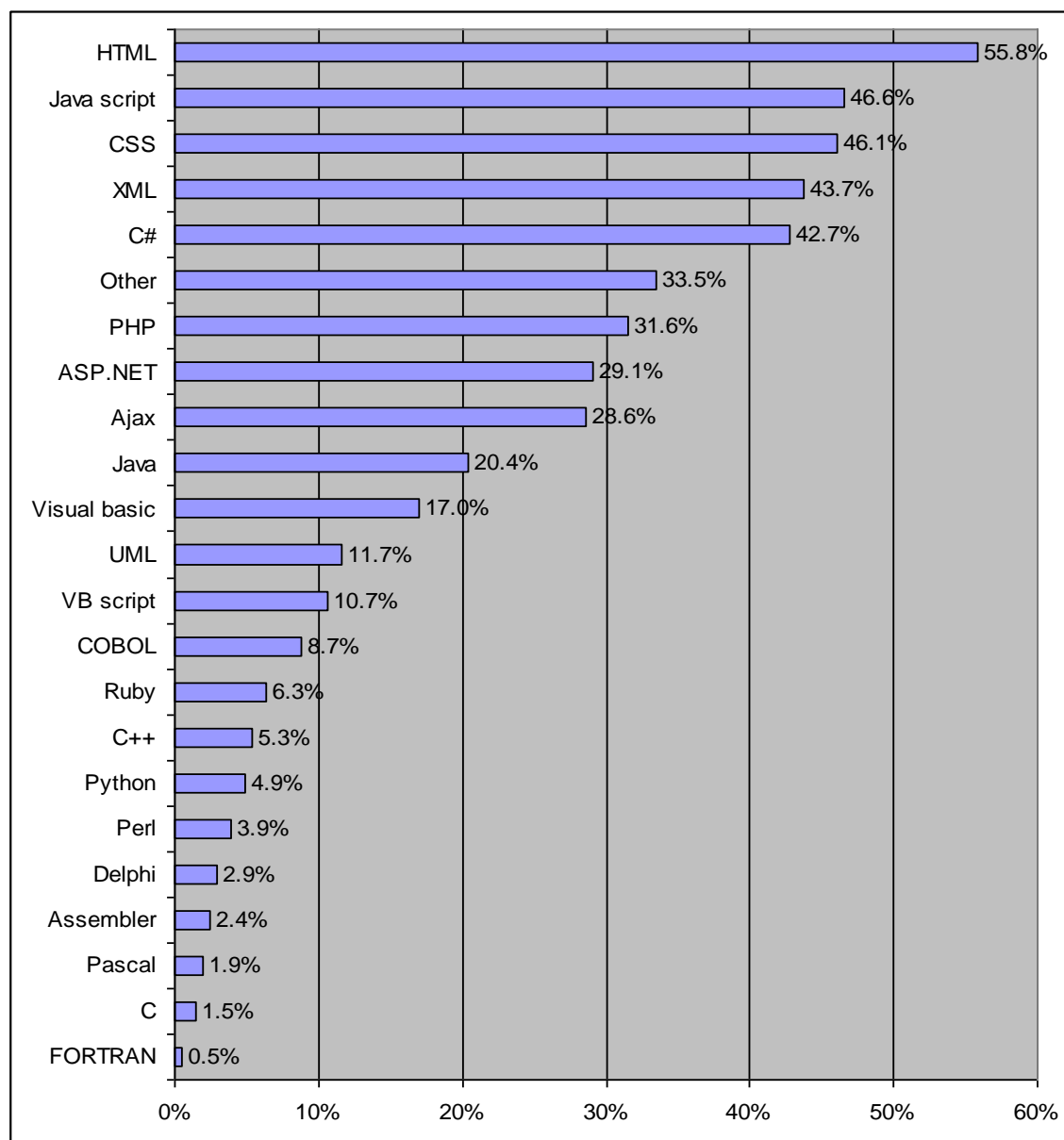


Figure 4.13: Programming technologies (n=206)

4.2.2.3 Database Management Systems

Question: What among these database management systems (DBMS) do you use most of the time?

Eight mainstream DBMS were identified and respondents were asked to indicate which DBMS are used while performing their tasks as software developers. With reference to Table 4.14 and Figure 4.14, there are five DBMS where each DBMS is used by more than 10 percent of respondents.

The most used top five DBMS are:

- i. Microsoft SQL server used by 53.9 percent of respondents
- ii. MySQL used by 48.1 percent of respondents
- iii. Oracle used by 22.3 percent of respondents
- iv. DB2 used by 12.1 percent of respondents
- v. Microsoft access used by 11.2 percent

However, 15 percent of respondents indicate to use other types of DBMS. These could be in-house specific DBMS and legacy flat files. Each of the following three DBMS: PostgreSQL, Sybase and Informix are used by less than 10 percent of respondents where Informix is the least used DBMS, used by 0.5 percent of respondents.

Table 4.14: Database management systems (n=206)

	Number of responses	Percentage
Microsoft SQL server	111	53.9%
MySQL	99	48.1%
Oracle	46	22.3%
Other	31	15.0%
DB2	25	12.1%
Microsoft access	23	11.2%
PostgreSQL	20	9.7%
Sybase	8	3.9%
Informix	1	0.5%

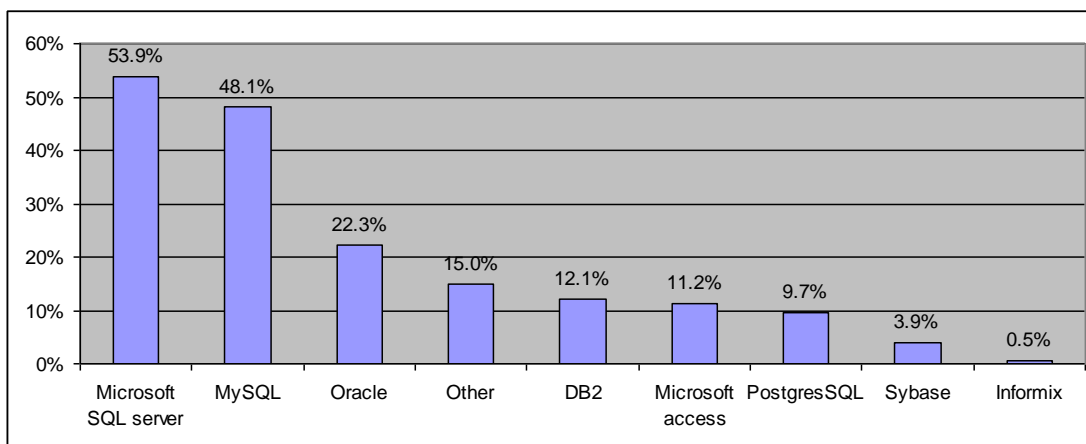


Figure 4.14: Database management system (n=206)

4.2.2.4 Code version control tools

Question: What among these code version control tools do you use most of the time?

Seven mainstream code version control tools were identified and respondents were asked to indicate which tools are being used while performing their tasks as software developers. With reference to Table 4.15 and Figure 4.15, there are three code version control tools where each tool is used by more than 10 percent of respondents. The most used top three code version control tools are:

- i. *Subversion used by 35.9 percent of respondents*
- ii. *Visual studio team system used by 23.8 percent of respondents*
- iii. *Visual source safe used by 18.9 percent of respondents*

However, 23.8 percent of respondents report not to use any code version control tool and 23.3 percent report to use other tools. Each of the following four code version control tools: Concurrent version system, Vault, Star Team and Perforce is used by less than 10 percent of respondents where Perforce is the least used tool, used by 0.5 percent of respondents.

Table 4.15: Code version control tool (n=206)

	Number of responses	Percentage
Subversion	74	35.9%
Visual studio team system	49	23.8%
None	49	23.8%
Other	48	23.3%
Visual source safe	39	18.9%
Concurrent version system	19	9.2%
Vault	4	1.9%
Star Team	3	1.5%
Perforce	1	0.5%

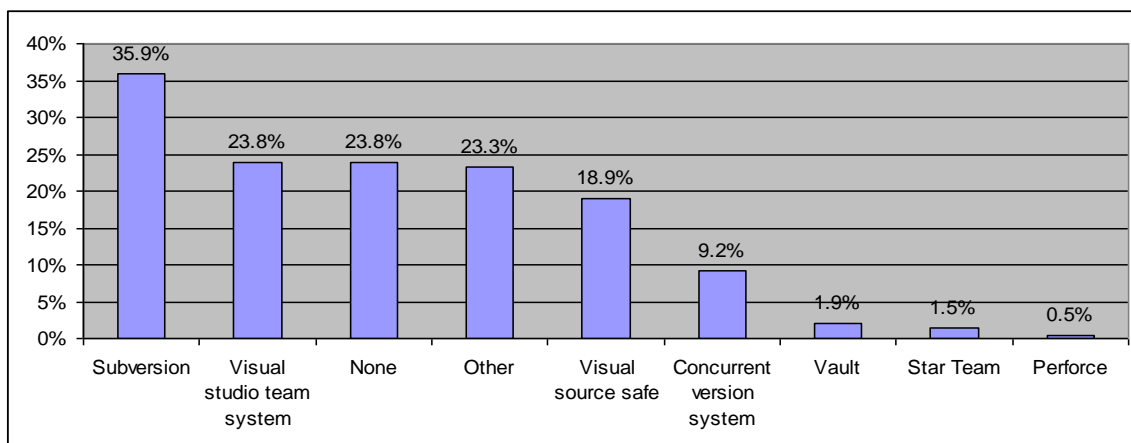


Figure 4.15: Figure 4.15: Code version control tool (n=206)

4.2.2.5 Application servers

Question: What among these application servers, do you use most of the time?

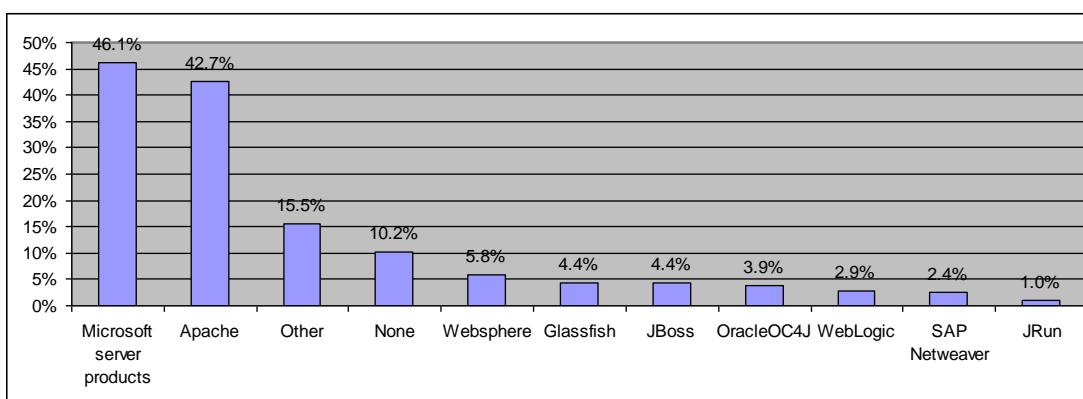
Nine mainstream application servers were identified and respondents were asked to indicate which application servers are mostly used at their respective work places. With reference to Table 4.16 and Figure 4.16 in the following section, there are two application servers where each application server is used by more than 40 percent of respondents. The most used top two application servers are:

- i. *Microsoft server products used by 46.1 percent of respondents*
- ii. *Apache used by 42.7 percent of respondents*

However, 10.2 percent of respondents indicate not to use any application server while 15.5 percent report to use other application servers. Each of the following seven application servers: Websphere, Glassfish, Jboss, OracleOC4J, WebLogic, SAP Netweaver and JRun is used by less than 10 percent of respondents where JRun is the least used server, used by 1 percent of respondents.

Table 4.16: Application server (n=206)

	Number of responses	Percentage
Microsoft server products	95	46.1%
Apache	88	42.7%
Other	32	15.5%
None	21	10.2%
Websphere	12	5.8%
Glassfish	9	4.4%
JBoss	9	4.4%
OracleOC4J	8	3.9%
WebLogic	6	2.9%
SAP Netweaver	5	2.4%
JRun	2	1.0%

**Figure 4.16: Application server (n=206)**

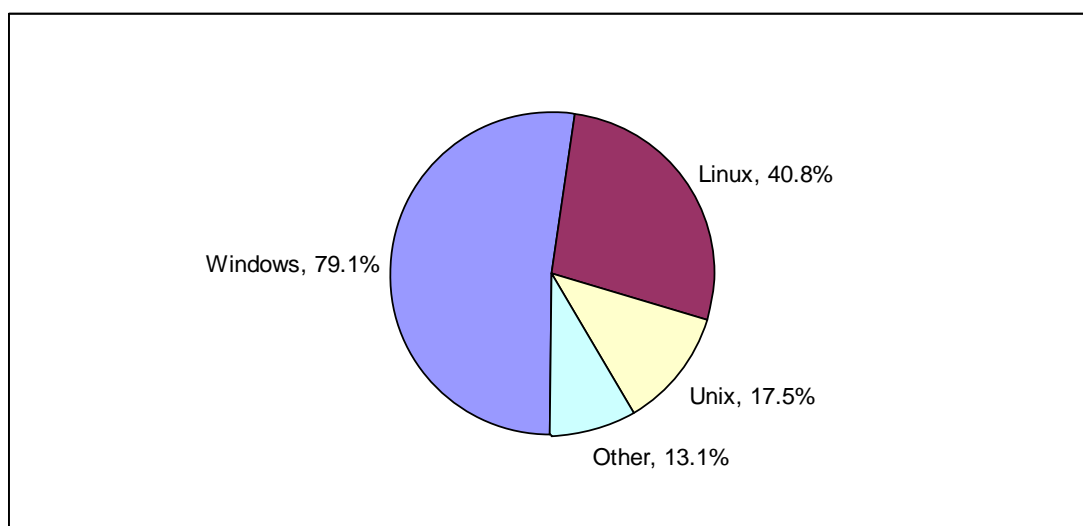
4.2.2.6 Software installation environment

Question: Among these environments, your software products are installed or running in which environment?

Three common software installation environments were identified and respondents were asked to indicate software environments where their software products are deployed. With reference to Table 4.17 and Figure 4.17 in the following section, majority 79.1 percent of respondents develop software systems for windows, 40.8 percent develop software systems for Linux and 7.5 percent develop software systems for Unix. Nevertheless, 13.1 percent report to develop software system for other deployment environment.

Table 4.17: Software installation environment (n=206)

	Number of responses	Percentage
Windows	163	79.1%
Linux	84	40.8%
Unix	36	17.5%
Other	27	13.1%

**Figure 4.17: Software installation environment (n=206)**

4.2.3 Tasks performed by software developers

This section presents tasks performed by software developers. Six common categories of tasks performed by software developers were identified. These categories of tasks cover the entire software development life cycle.

Question: Among these software development tasks, your software development tasks fall in which groups?

Respondents were asked to indicate categories of their tasks as software developers. With reference to Table 4.18 and Figure 4.18 in the following section, more than half of respondents perform all categories of tasks in the software development life cycle. Most of respondents, 88.8 percent, write codes for software systems. Eighty-two (82.5) percent perform tasks related to design of software solutions. 76.2 percent perform tests for software solutions. 72.8 percent perform

software maintenance. 68.9 percent analyse user requirements and 68 percent deploy software solutions.

Table 4.18: Software development tasks (n=206)

	Number of responses	Percentage
Analyse user requirements	142	68.9%
Design software solution	170	82.5%
Code software solution	183	88.8%
Test software solution	157	76.2%
Deploy software solution	140	68.0%
Perform software maintenance	150	72.8%

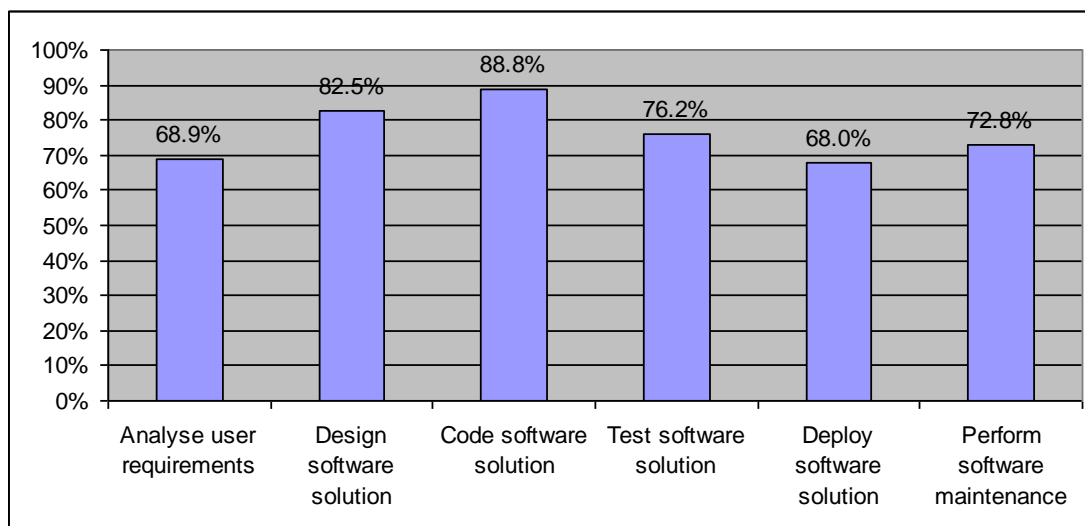


Figure 4.18: Software development tasks (n=206)

4.2.4 Skills of software developers

This section presents data related to fundamental skills required by software developers to develop software products. Two questions, 5-point likert scale question and an open-ended question were used to collect data regarding skills of software developers.

Question: While performing your tasks as a software developer, how well do you agree that the following skills are important for you to perform your tasks?

This question had ten 5-point likert scale sub-questions to address fundamental skills required by software developers. Respondents were asked to indicate how well they

agree with particular skills. For each sub-question, each respondent had to choose one of the given five choices: strongly agree, agree, don't know, strongly disagree and disagree. During data analysis, nevertheless, statistics for strongly agree and agree were combined together to form one "agree" group and data for strongly disagree and disagree were combined to form one "disagree" group. Hence, data collected were categorised into three groups namely agree, don't know and disagree.

With reference to Table 4.19 and Figure 4.19, three top most skills agreed to be important among software developers are:

- i. *Analytical and thinking skills; agreed by 96.6 percent of respondents*
- ii. *Communication skills; agreed by 94.7 percent of respondents*
- iii. *Ability to do self-study and research; agreed by 93.7 percent of respondents.*

Other skills that are agreed by more than 80 percent of respondents are: "Focus on customer needs" agreed by 91.7 percent, "Innovative while dealing with problems" agreed by 91.7 percent, "Team work" agreed by 88.8 percent, "Reuse of code" agreed by 86.4 percent and "Experience from previous work" agreed by 84.5 percent. The use of prototype is agreed by 66.5 percent while the least agreed skills are "Lack of ego" which is agreed by 65.5 percent of respondents.

Table 4.19: Skills of software developers - Agree (n=206)

	Agree	
	Count	Percent
Analytical and thinking skills	199	96.60%
Communication skills	195	94.70%
Ability to do self-study and research	193	93.70%
Focus on customer needs	189	91.70%
Innovative while dealing with problems	189	91.70%
Team work	183	88.80%
Reuse of code	178	86.40%
Experience from previous work	174	84.50%
Use of prototype	137	66.50%
Lack of ego	135	65.50%

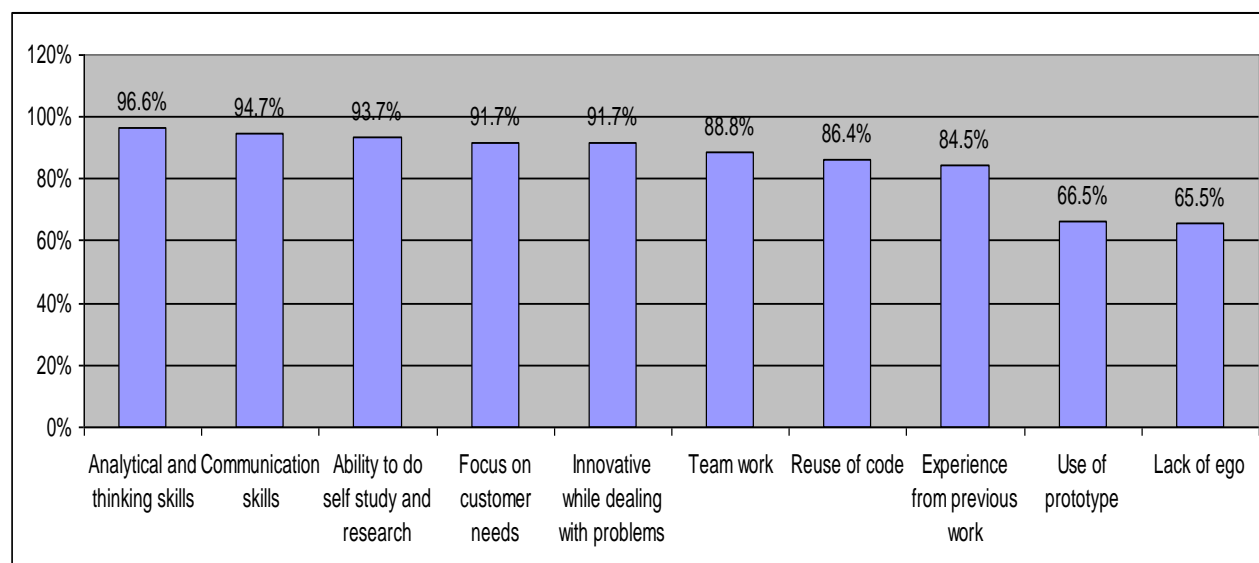


Figure 4.19: Skills of software developers – Agree (n=206)

With reference to Table 4.20 and Figure 4.20, two top most skills that are not known to be important among software developers are:

- i. *Use of prototype; not known by 22.8 percent of respondents.*
- ii. *Lack of ego; not know by 18 percent of respondents.*

For other skills, only less than 10 percent of respondents indicated not to know if those skills were important among software developers.

Table 4.20: Skills of software developers – Don't know (n=206)

	Don't know	
	Count	Percent
Use of prototype	47	22.80%
Lack of ego	37	18.00%
Experience from previous work	19	9.20%
Team work	13	6.30%
Reuse of code	13	6.30%
Focus on customer needs	11	5.30%
Innovative while dealing with problems	9	4.40%
Ability to do self-study and research	6	2.90%
Communication skills	5	2.40%
Analytical and thinking skills	4	1.90%

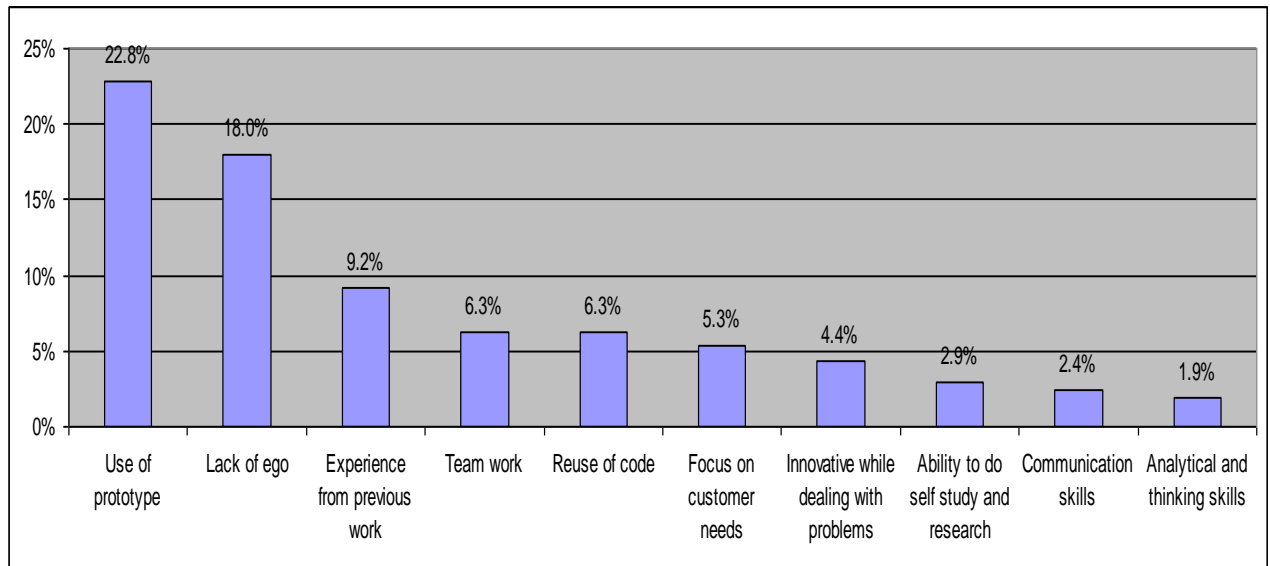


Figure 4.20: Skills of software developers – Don't know (n=206)

With reference to Table 4.21 and Figure 4.21, two top most skills disagreed by respondents are:

- i. *lack of ego disagreed by 16.5 percent of respondents*
- ii. *use of prototype disagreed by 10.7 percent of respondents*

For other skills, less than 10 percent of respondents indicated to disagree that those skills were important among software developers.

Table 4.21: Skills of software developers – Disagree (n=206)

	Disagree	
	Count	Percent
Lack of ego	34	16.50%
Use of prototype	22	10.70%
Reuse of code	15	7.30%
Experience from previous work	13	6.30%
Team work	10	4.90%
Innovative while dealing with problems	8	3.90%
Ability to do self-study and research	7	3.40%
Communication skills	6	2.90%
Focus on customer needs	6	2.90%
Analytical and thinking skills	3	1.50%

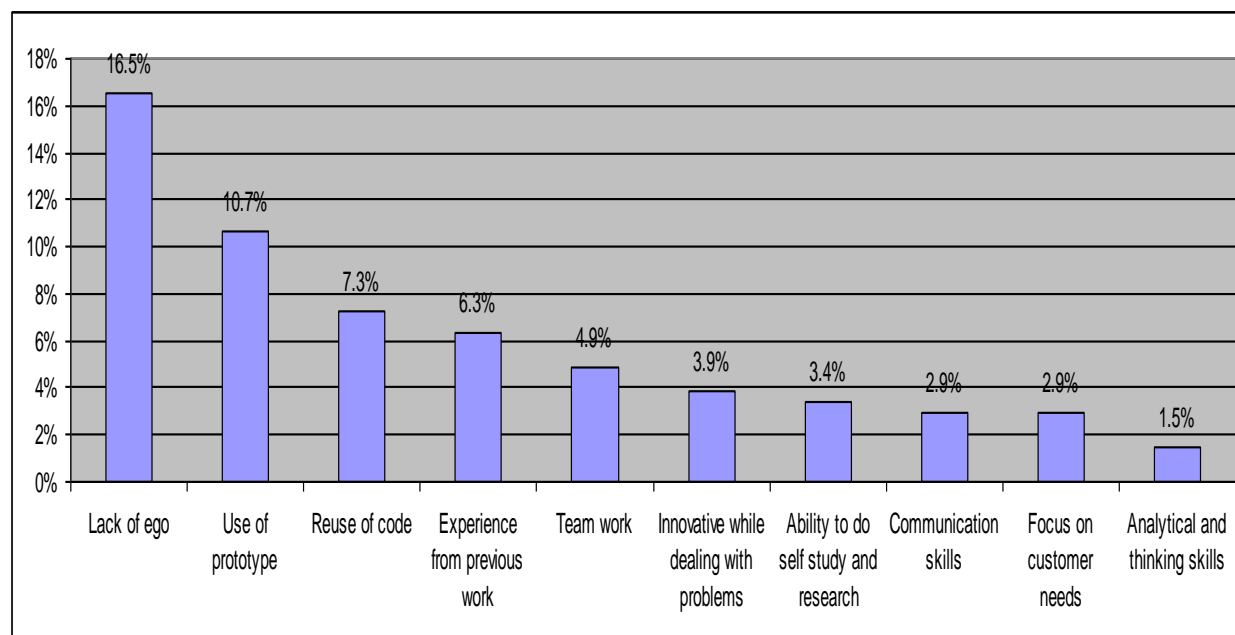


Figure 4.21: Skills of software developers – Disagree (n=206)

4.3 Qualitative data analysis

Qualitative data analysis was performed by using a content analysis technique. Content analysis, according to Berelson (1954:489), is a research technique for the objective, systematic and qualitative description of the manifest content of communication. Content analysis plays important roles when a researcher seeks to understand the content of a given series of text. According to Welman *et al.* (2005:221-222), content analysis can be described as a quantitative analysis of qualitative data. Qualitative data is a distinctive form of data namely language and texts (Gibbs, 2002:1). Welman *et al.* (2005:221) explains that basic techniques employed during content analysis involves counting of frequencies and sequencing of particular words, phrases or concepts in order to identify keywords, themes and patterns. Content analysis can be used to analyse personal documents, mass media material, open-ended questions of survey and unstructured interviews.

Turley and Bieman (1995) applied content analysis to analyse the text of interviews collected from software engineers. Brandt (2006:54) used content analysis during the study on perspective of IT⁹ teachers on the possible cost-effective means of networking computer facilities for schools in Grahamstown, in the Eastern Cape.

⁹ IT: Information Technology

Similarly Mlitwa (2009) used content analysis in his study on the factors affecting the usage and non-usage of learning management systems at tertiary institutions in the Western Cape. Content analysis is a quantitative data analysis of qualitative data (Welman *et al.*, 2005:221).

According to Weitzman and Miles (1995), there are several computerized software programs, which can be used to execute content data analysis. Such software programs include Atlasi and NUD-1st. In addition, Zelger and Oberprantacher (2002) describe about GABEK (Ganzheitliche Bewältigung Sprachlich Erfasster Komplexitat) as a computer-aided methodology that can be used for the content analysis of unstructured textual qualitative data from open-ended questions. Software packages such as Atlas.ti, N4 Classis, N5, N-Vivo and WinMax can be used to support content analysis of literature review (Di Gregorio, 2000:2). Content analysis can be performed by using N-Vivo, which is a tool for qualitative research data analysis (Welman *et al.*, 2005:224). According to Andrew *et al.* (2008:36), the use of N-Vivo not only has proved to be beneficial for the synthesis and analysis of mixed methods data but also N-Vivo enriches research findings. Ozkan (2004:594) reports that N-Vivo is a powerful tool that can facilitate sophisticated data coding, data searching, exploration of coded text and exploration of complex ideas. N-Vivo software provides facilities for data management, for coding and retrieving text and for theory testing (Crowley, Harré & Tagg, 2002:194).

While several computerized tools could have been used, following the fact that the amount of data collected could be managed with the use of advanced spreadsheet software, Excel 2010 was deemed suffices and used for qualitative data analysis during this research. Data from the open-ended question and interviews were collected then content analysis was performed manually by carefully reading text of data collected in order to identify patterns and themes. Therefore interviews were analysed using the content analysis technique as described above.

The following section present data analysis of open-ended questions of survey questionnaires and interviews performed.

4.3.1 Open-ended question (Other skills of software developers)

Question: What other skills do you possess that help you perform and accomplish your software development tasks?

This was an open-ended question where respondents were encouraged to provide as much information regarding other skills that they deem important to them as software developers. Data collected was analysed using content analysis technique. The researcher had to read all collected textual information in order to identify themes and patterns. Using Excel 2010 software, the researcher categorized collected information into patterns of skills where information related to one pattern of skills is placed into one group. The number of patterns of skills was calculated to determine frequency and percentage of all identified skills. With reference to Table 2.22 and Figure 4.22, eleven patterns of skills, some of which already appeared in the previous section, were identified as indicated in the following section.

Table 4.22: Other skills of software developers (n=154)

	Count	Percent
Technical skills	28	18.18%
Management skills	23	14.94%
Thinking skills	22	14.29%
Personal skills	19	12.34%
Users oriented	17	11.04%
Business skills	15	9.74%
Hard working	12	7.79%
Research skills	10	6.49%
Experience	4	2.60%
Communication skills	3	1.95%
Team work	1	0.65%
Total	154	100.00%

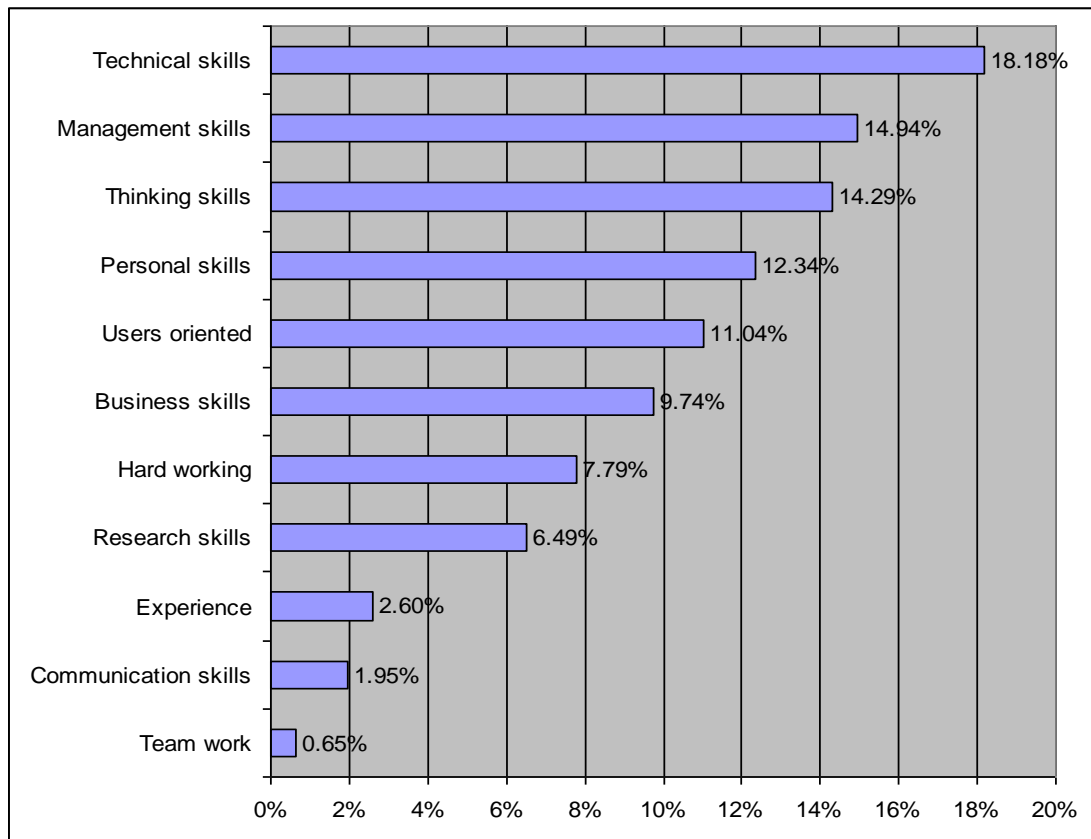


Figure 4.22: Other skills of software developers (n=154)

Patterns of skills reported by research respondents are:

i. Technical skills

Technical skills are rated the highest with 18.18 percent. Some of respondents reported about knowledge of software development frameworks such as MVC (Model View Controller) as critical to developing scalable, maintainable and testable software systems. ASP.NET MVC was cited as the common used framework for web applications. Other respondents mentioned about the use of design patterns and knowledge of web services as vital in their respective work places. Technical knowledge on developing service-oriented applications (SOA) was reported. In particular, knowledge of WCF (Windows Communication Foundation) was identified as critical for developing and deploying software services on the windows platform. Some of respondents reported that the management of source code in large software projects which normally have diverse and different teams in different geographical locations, could be challenging. Respondents reported that, in such collaborative software development projects, Team Foundation Server (TFS) is used for source control, data collection, reporting and project tracking. Hence, technical knowledge of TFS is critical while performing their tasks as software developers.

Other technical areas reported by respondents were knowledge of database design, sound understanding of relational databases and strong SQL background. This was deemed critical because most of software systems developed require some kind of database for data storage. According to Satzinger *et al.* (2004:487-524) many software systems require data to be created, stored, retrieve, modified and deleted, hence knowledge of database management systems is paramount. Respondents mentioned about Oracle, SQL server and MySQL as important database management systems. It was also mentioned that, according to software development best practice, software systems do not interact directly with database but via Object Relational Mapping (ORM) tools such as NHibernate, Hibernate, and Linq to SQL or Entity Framework. As such, some of respondents emphasized that knowledge of ORM is compulsory among software developers. Data synchronization between local applications and remote applications was mentioned. Respondents reported that knowledge of data synchronization technologies such RDA and merge replication is important especially in mobile applications.

Furthermore, respondents mentioned about HTML, CSS, JavaScript and jQuery. One of the respondent mentioned “*jQuery and JavaScript are the blood of web applications particularly for better user experience, this is a must know for software developers*”. Some of respondents reported about knowledge of XML and JSON as critical in developing web applications. This is because most of software systems developed are web applications; deployed and accessed via the Internet. Technical skills related to integration of software were mentioned. Respondents spoke about scenarios where one system had to exchange data with other systems and where new systems had to interact and communicate with legacy systems. The ability to be proficient and knowledgeable about several programming languages was also mentioned. Some respondents said that experience in using Object Oriented Programming (OOP) language such as C#, Java and C++ was important while other respondents reported that knowledge of Unix and Linux was critical to perform tasks in their respective software projects.

However, other respondents reported that working in large software projects demand more people skills than technical skills. Respondents explained that people skills imply skills that would enable a group of people to work together as a team. This includes teamwork, communication skills, negotiation skills and conflict resolution

techniques. One respondent, for example, reported that a software developer with fair technical skills and good people skills would be beneficial to large scaled software projects than a software developer with strong technical skills but poor people skills. This is because the success of large software projects depends on large interactions and collaborations of many and diverse categories of people such as users, managers and other software developers. The respondent, nonetheless, stressed that there is no substitute for technical skills; hence, a software developer with a good combination of technical skills and people skills would be ideal for large software projects.

ii. Management skills

Management skills were rated second highest with 14.94 percent. Respondents reported that the nature of their work is stressful, with tight deadlines and lots of pressure. One respondent said, “A software developer must be able to perform well under pressure”. Some of respondents mentioned about time management, planning and being proactive as critical to be able to handle pressure related to software projects.

iii. Thinking skills

Thinking skills were 14.29 percent were respondents reported that most of their tasks required critical thinking capability. Some of phrases used by respondents were creative thinking, logical thinking, excellent memory, analytical thinking, strong problem-solving skills and pattern thinking. For instance, one respondent said, “Life is made up of patterns, if you have the ability to find the patterns in things; it is a huge advantage as a software developer”.

iv. Personal skills

Personal skills were 12.34 percent. This group encompasses personal attributes that one needed in order to perform tasks as software developer. Respondents reported about health life style as critical to their tasks. Sports, martial arts and gym were cited by respondents. Other respondents reported about sense of humour, determination, pride, discipline and good attitude as important elements of software developers. One respondent said that “good attitude = gold”. Other respondents used phrases like “being awesome”, being social, being out going, being easy going and drinking coffee as important in order to work as a team in software projects.

v. Users oriented

User oriented skills were 11.04 percent where respondents indicated that users are very important to the success of software projects. As such, respondent mentioned that software developers must bear users in their minds in order to understand users' needs. Some of respondents mentioned about patience, tolerance of frustration with users, conflict management with users, persistence and perseverance. One respondent said, "It is important to keep users happy and never to give up".

vi. Business skills

Business skills were 9.74 percent where respondents used phrases like understand business environment, business analysis, business knowledge, understand business processes, problem solving skills and understanding business requirements. One respondent said, "A software developer is a business problem solver".

vii. Hard working

Hard working skills were 7.79 percent. Respondents used phrases like being hard worker, working even after normal work hours. One respondent said, "The ability to sit on my backside for extended period of time and putting in the inevitable overtime with a smile on my face is important". Other respondents said that software developers must be responsible, must have dedication towards work, must have commitment to quality of work and must have passion and interest in work.

viii. Research skills

Research skills were 6.49 percent where respondents reported that knowledge of using the Internet to search for solutions is critical. Google was mentioned as the most commonly search engine used by software developers. Some respondents reported to do research outside work hours. Some respondents reported to search for community forums and blogs in order to keep up to date with technology and share experiences. Blogs are web applications hosted on the website that allows users to post information, read information and add comments to posted information. According to respondents, the use of blogs has positive effects in bringing together people of similar interests. The same is reported by Pereira and Parker (2009:109) where blogs are used as an interactive means of communication by communities in the Western Cape Province. Other respondents reported about the need to learn new technologies regularly and learn fast. One respondent said, "Research skills to extract necessary information from loads of information in timely manner are critical".

One respondent mentioned, “Research skills are not copy and paste attitude practised by some of software developers”. The respondent discourages copy and paste attitude where software developers copy and paste codes for a quick fix of a given problem. The respondent said that one could not establish issues that would be introduced into the system by copying and pasting codes. The respondent hence encourages research skills where one has to find out about information that one does not know to obtain a solution, understand the solution and then implement an appropriate solution. According to one respondent, “most of business problems are similar and today’s problem might have been resolved yesterday”. “In order to avoid re-inventing the wheel, research skills are critical among software developers”, the respondent concluded.

ix. Experience

Experience was 2.6 percent. Some respondents reported that many years in software development projects played big roles in their present tasks. Some respondents reported experience comes with time but it can be enhanced with practice. One respondent said, “Students must interact with software development communities, while still at school, in order to gain a test of practical experience of real life software projects”.

x. Communication skills

Communication skills are 1.95 percent. Some of respondents reported about verbal and written communication skills being important to software developers. For instance, one respondent said that “the ability to communicate with ignorant (not stupid....ignorant in IT matters) without getting impatient” was critical.

xi. Team work

Lastly but not the least, it is teamwork with 0.65 percent where one respondent mentioned, “Knowledge sharing among team members in a given project is important”. It was also mentioned that most of software companies use wiki as source of depository for information sharing and technical assistance.

4.3.2 Data analysis of interviews

As indicated in chapter three, unstructured interviews were performed. Twelve software developers from six companies were interviewed. The interviews were performed on one to one basis where a researcher interviewed one individual after

another. All interviewees were software developers with working experience from five years, working as senior software developers. Each interview session took an average of one hour depending on the availability of interviewees. Figure 4.23 is a generic software solution indicating common technologies and tools used by interviewed software developers. Data collected during interviews is presented in the following section.

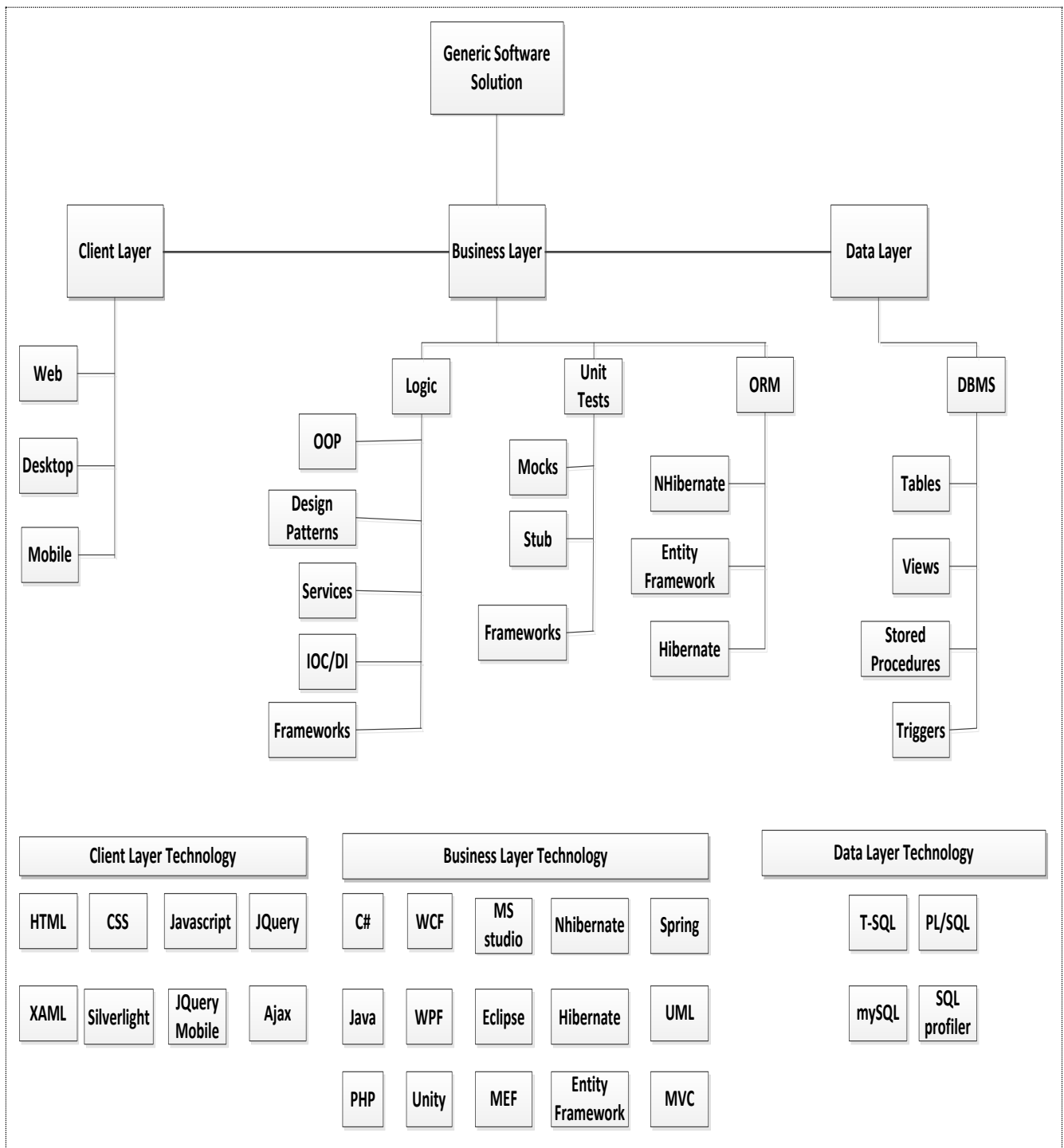


Figure 4.23: Technologies used by interviewed the software developers

4.3.2.1 First Company

At the time of interviews, the first company had seven software developers. According to interviewees, this small company can develop any software system requested by clients. The company has developed several desktop applications, web applications and mobile applications. The focus of the company is to develop sales force automation software and business intelligence applications running on mobile devices. During the interviews, it was also reported that the company has plans to launch a new project for developing mobile POS (Point Of Sale) software application primarily to be used by waiters for processing food orders in restaurants. Nonetheless, it was reported that there are limited resources of software developers with experience in software development for mobile devices.

The company is using Microsoft SQL server for data management. Programming languages used are C-Sharp (C#) and Java. According to interviewees, knowledge of C#, Java, T-SQL¹⁰, HTML, XML, JavaScript, JQuery and ASP.NET is critical for all web applications developed by this company. Most of Web applications were developed on ASP.NET platform using web forms. Nevertheless, it was reported that for new web projects, the company would be using ASP.NET MVC platform. It is because of advantages that ASP.NET MVC offers as opposed to ASP.NET web forms. Such advantages include support for development of testable, maintainable and scalable web applications. Hence, ASP.NET MVC is recommended for especially large software projects, software developers reported.

The company is developing mobile applications referred to as occasionally connected applications (OCA). Mobile devices host client applications, which continually request services via the Internet. However, if the internet connection goes off or if internet is not accessible, the client software in the mobile device has mechanism to tolerate poor Internet connection and continue to render critical services; hence the term “Occasionally Connected Applications”. It was reported that knowledge of developing software systems using services is critical to this company. All of software systems developed by this company utilize SOA (Services Oriented Architecture) using WCF (Windows Communication Foundation) framework to develop services oriented software components. Knowledge of WCF services was

¹⁰ T-SQL : Transact SQL; Structured Query Language for Microsoft SQL server database system

thus rated as compulsory for any software development work for this company. Software developers use SQLCE¹¹ and SQLite¹² for data management for mobile applications. The targeted deployment platforms for mobile applications are windows mobile, windows phone 7 and android. Therefore, software developers for this company need to be familiar with windows mobile, windows phone 7 and android operating systems. Data synchronization between client mobile applications and remote servers is critical in OCA mobile applications. Software developers reported that knowledge of data synchronization technologies such RDA (Remote Data Access) and merge replication are important skills. The company uses two IDEs; Visual Studio 2008 and Eclipse Helios. The company uses Subversion as code version control software.

Software developers reported that research skills are crucial for a software developer. One interviewee particularly reiterated the difference between research skills and copy and paste attitude practised by some of software developers. The interviewee discourages copy and paste attitude where software developers copy and paste software codes to fix the current problem while introducing other problems. He explained that with copy and paste attitude meaning that the software developer might have understood the problem but does not understand the solution. Hence, one cannot establish issues that would be introduced into the system by copying and pasting codes. Software developers encouraged research skills where one has to find out about information that one does not know and then obtain solution, understand the solution and then implement a correct solution appropriately.

According to interviewees, most of business problems are similar and hence present problem might have been resolved already, hence knowledge of software development patterns. In order to avoid re-inventing the wheel, research skill is critical among software developers, one interviewee emphasized. It was also mentioned that the more code one has to write, the more chances of introducing bugs in the software system. Hence, the approach of writing less code is preferred. This could be achieved by doing research on what has been done and is readily available to re-use instead of re-inventing the wheel and consequently introduce other software flaws. Google was reported to be the prefer choice of search engine

¹¹ SQLCE : Compact edition of structured query language used in mobile devices

¹² SQLite : Light version of structured query language used in mobile devices

and Mozilla Firefox was the preferred browser since it has built-in plugins such as fire bug, which is a built in tool used for debugging web applications. In particular, software developers reported to using Firebug plugin and chrome browser for debugging JavaScript, jQuery, CSS and inspecting the HTML/DOM¹³.

Software developers mentioned that, according to their tasks, they see and recognize two categories of skills; compulsory and subsidiary skills. They indicated that technical skills and research skills are compulsory for software developers to do their tasks. On the other hand, communication skills and business skills were considered important but subsidiary. They reported that those subsidiary skills are specific to company and hence given stronger technical background and research skills, a software developer can acquire business skills along the way. Nevertheless, they indicated that a good software developer should not only have technical skills and research skills but also business skills and communication skills.

It was also reported that a software developer should be someone with passion for software development. Time and skilled workers being scarce resources, the working environment of a software developer can be demanding and challenging especially for small software companies. Occasionally software developers have to put more time in their work. It was mentioned that there are times when interviewees work during weekends and often work from home after office hours. This is in order to meet project deadlines. As such, passion for software becomes the main driving force for software developers to work independently and without supervision in such time constrained working environment.

Teamwork was recognized as a key factor for project success. The company encourages team knowledge sharing where each software developer is given opportunity to present a topic on specific software technology of interest. Occasionally all software developers do participate in outdoor events such as a dinning together in restaurants and discuss vision and mission of the company as team. It was mentioned that teamwork is not practised at the expected degree as such the company is working on active team building strategies that will be executed regularly to promote the team spirit.

¹³ DOM : Document Object Model which explain how HTML elements fit together in web pages

In this company, all software developers perform all tasks from analysis of the problem to design of solution, coding the solution, testing and deployment of the solution. Therefore, it is required that software developers be fully versed with the Software Development Life Cycle (SDLC).

4.3.2.2 Second Company

The second company is a USA software company with software development team based in Cape Town. The Company is specializing in developing software products to process financial banking transactions. The company have clients worldwide. Most of clients are financial institutions, such as banks, co-operatives, retail business and insurance companies. Software projects executed by the company include developing software products to facilitate ATM (Automated Teller Machine) banking, Internet banking, mobile banking and point of sale systems (POS) financial transactions.

Most of software products are developed using Java using the OOP (Object Oriented Programming) approach to software development. Knowledge of solid concept of OOP such a class, interface, generics, composition, inheritance and aggregation were reported as fundamental to software developers for this company. In addition, some of software products are developed in Python. A software developer who is proficient in both Java and Python would be ideal for that company. The company use SQL server and DB2 as database management systems. Hence, knowledge of SQL and XML is essential technical skills. The company uses Eclipse Helios as the main IDE and Perforce as code version control software.

It was reported that the company embrace Test Driven Development (TDD) where every piece of code written must have unit tests and test cases to test all possible scenarios of a given unit of code. During maintenance, unit tests are executed and all tests must successfully pass prior maintenance. After maintenance, tests are executed again to ensure all tests pass successfully. This is because fixing one problem could introduce other software flaws commonly known as bugs. This has been the case especially to large software systems when TDD is not enforced. As such TDD comes to rescue where software developers using test harness and test cases are able to confirm that the software system is still behaving as expected and no bugs were introduced to the system during maintenance.

All software projects are big and hence are developed and supported by a group of skilled software developers. There are about five teams where each team has ten to fifteen software developers. Working in such an environment demands not only technical skills but also people skills. This is because there are lot of communications and interactions among software developers in teams of large magnitude. Such meetings include software sprint planning sessions. It was also indicated that a software developer with fair technical skills and good people skills would be beneficial to such a company than a software developer with strong technical skills but poor people skills. It was also mentioned that there is no substitute to technical skills. Hence, a software developer with good combination of technical skills and people skills would be ideal for such a company. Software developers explained that people skills implies any skills that would enable a group of people to work together as a team. This includes teamwork, communication skills, negotiation skills and conflict resolution techniques.

Some of software developers mentioned that technical skills are critical to software developers at a junior level. This is because tasks assigned to junior developers are clearly defined and hence junior developers would concentrate on developing and writing codes to provide the already identified solution. This gives junior developers an opportunity to strengthen their technical skills while acquiring people skills. Hence, junior developers have less interaction with people since mostly they are limited to interacting with individuals in the same project team. On the other hand, senior developers are expected to interact with diverse categories of people including users, managers and other software developers. Senior developers have some kind of management role to play hence people skills becomes critical. One software developer mentioned that people skills come with lots of experience; hence, a software developer with strong technical skills, with guidance, could develop people skills. It was identified that software developers consider technical skills as a priority to people skills.

However, while both technical and people skills are considered critical skills among software developers, the company gives priority to technical skills and recognizes that an individual with strong technical background is suitable for a software development job than a person with poor technical skills but strong people skills.

This company recognize teamwork as a key success factor in software projects. Interviewees reported that collaboration with other software developers within a team is essential. This is because success of a project is regarded as an output of team efforts. As such, the company has regular team building sessions where software developers from all teams took part and actively participate in sport events such as bowling ball and paintball. In addition, there is a weekly session geared at bring all software developers together. During this session, the company provides food and drinks for all software developers to eat together in the same canteen and share ideas on work and none-work related matters. This is an attempt to promote the culture of knowledge sharing and team spirit.

In this company, software developers perform all tasks from analysis of the problem to design of solution, coding the solution, testing and deployment of the solution. The company demand that all software developers be fully versed with the Software Development Life Cycle (SDLC).

Interviewees, however, indicated their dissatisfaction towards work performance of new employees who had qualifications from tertiary institutions but failed to execute basic tasks as software developers. It was indicated that most of these new employees have to learn lots of technologies and concepts from the scratch. This makes their learning curve steep and long; hence costly to the company. Some of them voluntarily resigned their jobs as software developers because they could not handle pressure following limited technical background.

4.3.2.3 Third Company

This is a software company whose focus is to provide consulting services and custom software development in the Western Cape Province. The company is mainly involved in custom development of web applications, mobile applications and desktop applications. Most of its clients are insurance companies, health and fitness institutions, financial institutions and retail business companies. Software projects executed by the company include developing software products to facilitate time management, data management for health institutions, software to process banking transactions and software for data management of fitness centres.

Most of software products developed target the windows platform. Hence, the company focus on utilizing Microsoft technologies to develop software systems. In

order to cater for clients who use Microsoft technologies, C# and SQL server are the main programming technologies utilized by the company. A software developer who is proficient in C# programming language and T-SQL is an ideal candidate to work for this company. The company embrace OOP approach to software development. As such, solid knowledge of OOP such as class, interface, inheritance, composition, aggregation is essential.

Interviewees indicated that knowledge of technical skills is critical while performing their tasks as software developers. They said, after all, their major task is to write codes in order to solve business problems. It was found that technical skills for software developers for this company are proficient in C# programming language using OOP approach, T-SQL, JavaScript, jQuery, HTML, XAML, and CSS. Software systems developed by this company are based on SOA using WCF services framework. The company uses NHibernate and Entity Framework (EF) as the major Object Relational Mapping tools. It was emphasized that knowledge of design patterns and testing frameworks were essential. In order to improve performance of software, software developers mentioned about SQL profiler and NH profiler as important tools used to analyse and understand codes executed by the database engine. Using these tool one is able to identify performance issues and take necessary precautions to improve software performance. The company uses Visual studio 2010 as the main IDE and TFS (Team Foundation Server) as code version control tool.

Most of web applications developed by this company employ memcache; a high-performance distributed memory object caching system, as a caching mechanism in order to improve software performance. This caching mechanism is implemented such that when users send new request, response is fetched from database and stored in a memory object such that when the same request is executed, the response is retrieved from memory, which is faster and hence improved user experiences. Hence, knowledge of memcache is important in this company.

Pattern thinking as problem solving skills was recognized by interviewees. It was reported that software developers should be able to identify patterns and use patterns to solve problems. Patterns are identified as best practice knowledge that could solve most of re-occurring business problems. It was reported that knowledge of patterns enables software developer to write pieces of codes that are

maintainable, scalable, testable and extendable. Interviewees reported that some of basic design patterns commonly used by this company are singleton, factory, façade, mediator, adapter, decorator, event aggregator, service locator, publish-subscribe pattern and Inversion of Control (IoC) pattern. Therefore, knowledge of patterns is fundamental to software development in this company.

Interviewees reported that by considering patterns and implementing design patterns appropriately, it does enforce the five critical OOP principles as indicated by the famous acronym “S.O.L.I.D” principles (Martin & Martin, 2006). These five principles are essential to writing good quality, maintainable, testable and extendable pieces of software components. Each letter in the S.O.L.I.D acronym refers to another acronym representing key principles of OOP as elaborated below:

i. SRP: Single Responsibility Principle

This refers to a situation where any class or unit of code should focus on doing one thing or have one responsibility only and hence one reason to change.

ii. OCP: Open Closed Principle

This refers to a situation where any class or unit of code should be open for extension but closed for modification. Therefore, one should be able to extend class behaviour without modifying the class. This could be achieved primarily by using inheritance and composition.

iii. LSP: Liskov Substitution Principle

This refers to a situation where Subtypes must be substitutable for their base types. Hence subclasses should behave nicely when used in place of their base class.

iv. ISP: Interface Segregation Principle

This refers to a situation that clients should not be forced to depend upon interfaces that they do not use. It advocates that software developers should develop fine-grained interfaces that are client specific, instead of fat interfaces that are not cohesive.

v. *DIP: Dependency Inversion Principle*

- a. *High-level modules should not depend on low-level modules. Both should depend on abstractions.*
- b. *Abstractions should not depend upon details. Details should depend upon abstractions.*

This suggests that one should use lots of interfaces and abstractions to decouple dependency among software entities.

While some of web applications are written on ASP.NET web forms platform, it was indicated that all new web application are developed based on MVC framework. It was found that the company employs the MVC architecture, in particular ASP.NET MVC framework, to develop new software systems. It was reported that knowledge of MVC is important for software developers working for this company.

The company is also involved in development of mobi-sites which are websites accessed via mobile devices using mobile browsers. It was reported that jQuery mobile which is a client side technology for design of user interface and its behaviour and HTML5 are critical. Moreover, the company embraces Test Driven Development (TDD) as such software developers are required to write unit tests for every unit of code written. It is mentioned that an individual versed with TDD has greater chance of working for this company.

For desktop and standalone applications, software developers utilize WPF (Windows Presentation Framework), which is a unified programming model for developing standalone rich client user experiences, engineered by Microsoft Company. The framework employs XAML¹⁴, which is XML based programming technology for designing user interfaces. Software developers use MVVM (Model View ViewModel) as a programming pattern for WPF applications. Knowledge of Managed Extensibility Framework (MEF) was identified as important for software developers working for projects that require high degree of scalability and extensibility. MEF technology is a framework that allows software developers to develop different software components and plug them together at run time a technique referred to as composition. The framework makes software system easily extendable and scalable in order to include other independent functionalities, one interviewee reported.

¹⁴ XAML : Extensible Application Markup Language

Software developers mentioned of communication skills as being important while executing their tasks. This is because all software developers are required to communicate among each other within a team and often they are required to participate in project planning, carry out Proof of Concept (POC) sessions and perform project demonstrations to clients, management and other team members.

In addition, the company embraces agile methodology of software development, as such; each project team was reported to have a daily meeting at 10 AM for fifteen to twenty five minutes. This is known as “stand-up” simply because the meeting is executed while all members are standing up as opposed to sitting down in order to discourage prolonged and unnecessary discussions. This is a brief meeting to ensure that every member in the team has something to do. It is during this meeting where every software developer briefly explains to the team on what is currently working on and possibly report on problems, if any. This is a typical scenario where communication skills play important roles.

Teamwork was recognized as key for project success. This is because success of a software project is regarded as an output of team efforts. As such, the company has regular team building sessions where software developers from all project teams took part and actively participate in sport events such as bowling ball, karting, cricket and paintball. In addition, the company has a monthly event geared at bringing all employees together. During this event, the company provides food, drinks and sport facilities for all employees to participate. During this event, new employees are introduced, officially welcomed and encouraged to be part of the team. More over individuals who went extra mile in performing their tasks for the sake of the team are nominated and recognized. Then one of the nominated employees is issued certificate of recognition as the employee of the month and rewarded a cash voucher and material presents. This is in order to encourage and promote the team spirit based on the premise that each individual would like to be the employee of the month, nevertheless to be an employee of the month one needs to exhibit extra efforts in performing tasks for the success of the team.

According to software developers of this company, technical skills and communication skills were reported as being critical while executing their tasks for the success of their software projects.

It was found that all software developers perform all tasks; from analysis of the problem to design of solution, coding the solution, testing and deployment of the solution. The company enforces that all software developers be fully versed with all tasks performed in the Software Development Life Cycle (SDLC).

It is worth to mention that interviewees from this company reported about technical weakness in most of new software developers recruited in recent years. Interviewees said that their company has long relationships with tertiary institutions. As such, the company provides an internship programme and employs many new candidates directly from tertiary institutions. “Nevertheless there is a big difference in technical ability between recently employed software developers and those of olden days (say prior 2008), software developers employed prior 2008 had reasonable technical knowledge to work as software developers but new software developers; employed after 2008, have very limited technical skills to function as software developers”, one interviewee mentioned. It was reported that most of new software developers failed to handle very basic tasks that an individual with a three years diploma in software development is expected to do. They have a long learning curve and it takes long time before they become productive to the company. This is because most of them have to learn most of software technologies from the scratch. This is contrary to olden ages where after internship; interns would be ready to embark on software projects with confidence, one interviewee concluded.

Furthermore, it was reported that there are few local software developers with experience in developing software applications for mobile devices. Interviewees indicated that the company has vision to invest in mobile software projects; nevertheless, there is a lack of experienced software developers with technical skills pertaining to android software development and iPhone software development. In general, there are limited skills on software development for mobile devices.

4.3.2.4 Fourth Company

The fourth company is a business organization specializing in online media. The company has several websites used for hosting media related matters such as daily news and online magazines. The main software project for this company focuses on using Web technologies to develop and support websites used for hosting media related matters and online magazines. Other software development work performed is developing software tools used internally to support the company business. This

includes software tools used to upload, download and manage data for websites and other online portals. The company has its own software developers whose main task is to support and maintain current websites. However, for big projects the company utilizes services offered by other software consulting agencies. The consulted software developers normally work for the duration of the project after which they would go back to their respective companies when projects are completed.

Technologies used by this company could be categorized into client side and server side technologies. Client technology refers to any technology that functions on the browser such as rendering and animations of web pages. Software developers reported that knowledge of HTML, JavaScript, JQuery and CSS were essential. With regard to server side technology which is technology that function on the server machine, it was reported that proficient in C# programming language is critical. In order to be productive a software developer must have solid background of OOP (Object Oriented Programming) and design patterns. The company use SQL server 2008 as a database management system hence knowledge of T-SQL is important. It was also identified that the data layer of systems developed utilizes NHibernate as the ORM tool. It was also indicated that knowledge of NHibernate is important. The company uses Visual studio 2010 as the main IDEs and TFS (Team Foundation Server) as code version control software.

Teamwork was mentioned. Software developers reported that collaboration with other software developers within a team is essential. Again, it is because success of a project is regarded as an output of team efforts. As such, the company has regular team building sessions where software developers from all teams took part and actively participate in sport events such as bowling ball, karting, paintball and cricket. The company also has other social events aiming at bringing all software developers of a given team together. During these events, all software developers are offered a compulsory invitation to meet, eat and drink together as a team. Again, this is an attempt to encourage the culture of knowledge sharing and team spirit.

Mostly software developers work in team of about ten people. Software developers are required to attend meetings with business owners, perform demonstrations and report to project managers. It was therefore reported that communication skills is important to software developers working for this company.

The major task performed by software developers is maintenance of previous work and integrating new features requested by clients. Software developers mostly perform tasks related to coding and testing software solutions.

4.3.2.5 Fifth Company

The fifth company is a business organization also specializing in online media. The company however specialized on developing interactive social website and software focusing on tourism centred information. This is an attempt to address the problem of inconsistency of information provided by tour guides as it is reported that often there seem to be inconsistency in information provided by tour guides where people gives information as they know it which is not necessarily correct. Thus, the company is exploring this opportunity by using software systems as virtual tour guides. In order to utilize the social website, one needs to be registered as a member. Once registered the software grants members rights and privileges to upload and download data and share it among members. The company develop software that alert tourists to the point of interest close to them as they drive in a particular area. The software is installed in a specific device and plugged into the car. As the car drives, the software picks up signals of point of interests and using a voice detection tool, the software instructs and informs tourists of what is happening in the area. The software acts as a virtual tour guide informing tourists of point of interest in the area such as game reserves, mountains, museums, archives, monuments, art centres to mention but a few. The company also provide mobile version of the above-mentioned social website for all major version of mobile device; android, mobile windows 7, Blackberry and iPhone.

Technologies used for software development are HTML, JavaScript, JQuery, CSS, C# and Java. For database management system, the company use SQL server 2008 and Oracle. It was reported that knowledge C# and Java is critical for software systems developed by this company. The company uses two IDEs; Visual Studio 2008 and Eclipse Helios.

It was identified that the critical skills required by this company are technical skills. Software developers must be proficient particularly in C# and Java. The company tends to outsource software developers from India and Ukraine. According to interviewees, the company had to resort to outsourcing software developers from abroad, simply because, local software developers have limited technical knowledge

of software development for mobile devices and those few with expertise are very expensive. It was also reported that a few local software developers resigned their job because they could not handle their software development tasks following limited technical background on software development for mobile devices.

The company has software developers working in diverse team across geographical borders. There are regular contacts between software developers in one country to another country, via telephone and Skype technologies. Software developers are required to perform demonstrations and report to project managers. As such, communication skills were identified as important for software developers working for this company.

The major task performed by software developers is maintenance of previous work and integrating new features requested by clients. Software developers mostly perform tasks related to coding and testing the software solutions.

4.3.2.6 Sixth Company

This is the last software company to participate in interviews during this research. The company focuses on providing software consulting services and custom software development. The company provides its services in South Africa, Europe and partly in Asia. The company is mainly involved in custom development of web applications, mobile applications and desktop applications. Most of its clients are insurance companies, health institutions, financial institutions and retail business companies. Most of software products developed target Microsoft technologies (.NET) and Java EE platform. Hence, the major programming languages utilized by the company are C# and Java. A software developer who is proficient in C# and Java is an ideal candidate to work for that company. The company uses Visual studio 2010 and Eclipse Helios as the main IDEs. TFS (Team Foundation Server) and Subversion are the main code version control software.

In this company, it was indicated that knowledge of technical skills is critical for software developers to perform their tasks. Software developers indicated that their major task is to design software and write codes in order to solve business problems. It was found that technical skills for software developers for this company are

proficient in C#, Java, T-SQL, PL-SQL¹⁵, JavaScript, JQuery, HTML, XAML, and CSS. The company uses NHibernate and Hibernate as the major Object Relational Mapping tools. Interviewees mentioned that knowledge of design patterns and testing frameworks are essential. For performance, knowledge of SQL profiler and NH profiler were mentioned. Using these tool one is able to identify performance issues and take necessary precautions to improve software performance. It was also found that the company employs the MVC architecture to develop software systems. It was reported that knowledge of MVC is important for software developers working for this company. Moreover, the company embraces Test Driven Development (TDD) where software developers are required to write unit tests for every unit of code written.

For desktop and standalone applications, software developers utilize WPF (Windows Presentation Framework). As already mentioned, this is a unified programming model for the development of standalone software that has rich client user experiences. In order to benefit from this programming model, it was reported that software developers must understand XAML and the implementation of MVVM (Model View ViewModel) as a programming pattern for the WPF framework.

Software developers mentioned of communication skills as being important while executing their tasks. The company employs agile methodology for software development. As far as communication is concerned, there is a daily stand-up meeting where each software developer briefly report to the team on the status of tasks being performed. This is a brief meeting of fifteen to twenty five minutes performed while all members are standing-up as opposed to sitting down. It is during this session where each software developer is allocated new tasks. Hence, communication skills are important since there are always interactions of software developers within and outside project teams. Software developers are required to communicate among each other within and outside a team and often they are required to perform project demonstrations to clients, management and other team members. According to interviewees from this company, technical skills and communication skills were reported as being critical while executing their tasks.

Teamwork was recognized as critical for project success. Software developers reported that collaboration with other software developers within a team is essential.

¹⁵ PL/SQL : Structured Query Language for Oracle Relational Database

This is based on grounds that success of a software project is deeply rooted on team efforts. As such, the company has regular team building events where software developers from all project teams took part and actively participate in sports event such as bowling ball, karting, and paintball. Again, this is an attempt to encourage the culture of working together as a team.

Software developers perform all tasks from analysis of the problem to design of solution, coding the solution, testing and deployment of the solution. All software developers are required and expected to be proficient and fully versed with the Software Development Life Cycle (SDLC).

4.4 Chapter Summary

This chapter presented data analysis of research data collected via survey questionnaires and interviews. Data presented is about the profile of software developers, software development tools, tasks performed by software developers and skills required by software developers to perform their tasks well.

4.5 Conclusion

Seventy eight percent of questionnaires (Table 4.1) were thoroughly completed and thus used for this study. Coupled with data collected via interviews, the triangulation methodology was successful and appropriate for the study. Based on data presented in this chapter, knowledge of software development tools, tasks performed by software developers and skills required by software developers remain critical for the success of software projects.

The following chapter five focuses on research discussion.

CHAPTER FIVE: RESEARCH DISCUSSION

“Always code as if the guy who ends up maintaining your code will be a violent psychopath who knows where you live”
(Martin Golding)

5.1 Introduction

This chapter discusses data collected during the research. The discussion is based on results of quantitative data analysis and qualitative data analysis as presented in chapter four. Most importantly, the chapter leads to recommendations provided in chapter six geared towards improving competencies of software developers.

Discussion on results of quantitative data analysis and qualitative data analysis is presented in section 5.2 and section 5.3 respectively. This discussion is based on analysis of data collected via survey questionnaires and interviews. The chapter concludes with chapter summary in section 5.4 and conclusion in section 5.5.

5.2 Discussion on results of quantitative data analysis

As indicated in chapter four, quantitative data analysis was performed on closed-ended questions of survey questionnaires. This section discusses results of data collected via closed-ended questions of questionnaires. This data reflects the profile of software developers who participated in this research. This data also reveals software development tools, tasks performed by software developers and skills required by software developers.

5.2.1 Profile of software developers**5.2.1.1 Gender of respondents**

In a democratic country where gender equality is considered as a human right, it leaves a lot to be desired if the software development industry continues to be male-dominated. Both the industry and institutions have a role to play to boost up the number of female software developers. This is in order to resolve gender imbalances within the software development industry. With reference to Table 4.2, the majority 80.6 percent of respondents are male and only 19.4 percent of respondents are female. The ratio of males to females is 4-to-1. The ratio is close to findings reported

by Turley and Bieman (1995:21) in their study on competencies of software Engineers in the USA were the male to female ratio was reported to be 3-to-1. It is thus recommended that institutions increase the intake of female learners to undertake software development studies. Similarly, the industry should encourage females with appropriate skills to assume different positions as software developers.

5.2.1.2 Age of respondents

As the Internet and information technology advance the need of software developers will continue to increase following the proliferation of software projects. With reference to Table 4.3, the current work force for software development is between 21 years to 49 years. This group makes 93.7 percent (68.9 plus 24.8) of total respondents. While most of respondent acquired software development skills at tertiary education, software development studies should be introduced in high schools. This will bring awareness among our communities and encourage learners to study software development during their tertiary education. The industry could also contribute by visiting high schools and speaking to high school learners about software development and carrier path. By doing so the Western Cape Province will always have a pool of young software developers available to successfully execute software projects.

5.2.1.3 Race of respondents

South Africa population is divided into four major races; black, white, coloured and Indian (Bornman, 2006:387). In a democratic country where racial imbalances are discouraged, the current race ratio of software developers deserves attention. With reference to table 4.4, the majority 65 percent of respondents are white, 14.1 percent are black, 11.2 percent are coloured and 7.8 percent are Indian. This indicates that the software development industry in the Western Cape Province is white dominated. While there are many factors that could be attributed to the lower number of software developers of other races, the challenge is to the industry and institutions to bring about racial equilibrium in the software development industry. Institutions should encourage learners from other races to undertake software development studies. Likewise, the industry should ensure that people of other races who have appropriate skills assume positions as software developers.

5.2.1.4 Citizenship of respondents

With reference to Table 4.5, the majority 88.8 percent of respondents are South Africans. However, for non-south African respondents the majority 5.3 percent of respondents are from Asia. This indicates that software developers from Asia are preferred than those from other African countries and other continents. This could be attributed to the fact that software developers from Asia have required skills and hence on demand.

Nonetheless, knowing that the current digital divide affecting the African continent is fundamentally rooted from poor education and lack of ICT infrastructure, the industry has role to play in combating the digital divide in Africa. While South African institutions accept and allow foreign students from other African countries to study software development courses, the industry should consider providing practical experience to these students from other African countries. The industry should at least accept foreign students for internship programmes and temporary employment in order to share knowledge and practical experience for the betterment of Africa as a continent. This is very much important because, during the research, it was found that most of companies have the policy of not accepting students from other African countries for internship and temporary employment. This is due to perception that foreign students are viewed as a negative resource; would leave the country and hence add no value to South African companies. Nonetheless, considering the proliferation of Internet technology and the expansion of the digital village, ICT goes beyond geographical borders. Likewise, the benefit of software developers goes beyond geographical borders. For instance, it was found that some of South African business companies have software developers based in Czech Republic, Germany and India. Moreover there are South African business companies in other African countries. Hence, geographical borders should not be used against future software developers for Africa as one continent. Accepting African foreign students for internship and employment will increase the number of knowledgeable and skilled software developers available to undertake software projects in Africa, the continent where digital divide is rampant.

5.2.1.5 Work experience of respondents

Western Cape Province has a pool of knowledgeable and experienced software developers. With reference to Table 4.6, respondents with work experience between four years to over ten years are 69.9 percent (22.3 plus 17 plus 30.6). Knowledge and experience from these experienced software developers should be preserved and disseminated among the software development industry and community at large. Institutions should connect with these experienced software developers and allow them to be part of the curriculum development for software development studies. This will enable institutions to offer education that address industry needs. The industry has role to play in providing practical experience to institutions. The industry should therefore allow experienced software developers to regularly visit institutions and speak to learners about real life software projects and share their practical experiences. The researcher does suggest the need of localized software development forums and online communities where students and experience software developers could share practical experiences regarding software projects.

5.2.1.6 Time taken to become self-dependent software developer

Some institutions do offer internship programmes where learners are given opportunity to work in the industry and gain practical experience. With reference to Table 4.7, 68 percent (37.9 plus 30.1) of respondents indicate that it took them between six to twelve months to be able to perform their tasks independently as software developers. Therefore, six month to one year could be considered as ideal for a novice software developer to become experienced with software development activities. It is thus encouraged that learners participate in internship programmes for a minimum of six months. However, it is the responsibility of both the institution and the industry to ensure that the internship programme is well utilized for learners to acquire solid practical experience and be able to work with minimum assistance from other software developers.

5.2.1.7 Software development certification status

With reference to Table 4.8, only 28.6 percent of respondents are certified software developers, the majority 71.4 percent are not certified. This could be attributed to the fact that certification training is not compulsory if a software developer has basic education pertaining to software development. Nonetheless, certification courses

remain an integral part of software development education. Most of software companies encourage software developers to undertake certification courses. Hence, institutions should work together with certification centres in order to provide better education required to meet the industry demands.

5.2.1.8 Education background of respondents

With reference to Table 4.9, the majority 72.9 percent (35 plus 37.9) indicates to have diploma and bachelor degree. Thus, diploma and bachelor degree certificates form the basic education foundation of software developers in the Western Cape Province. However, most of software companies consider first degree as the minimum requirement for a software developer to qualify for employment. Therefore institutions has role to play in order to ensure that learners qualify with at least first degree certificates. Most importantly, institutions and the industry must work together in this respect to ensure that the degree programmes offered cater for the industry demands and not just another qualification.

5.2.1.9 Current position of respondents

With reference to Table 4.10, middle level and senior level software developers form 88.6 percent (25.7 plus 52.9) of respondents. This indicates that the Western Cape Province has a large pool of knowledgeable and experienced software developers. Knowledge and experience from these software developers must be preserved and utilized to expand the body of knowledge of software development in the Western Cape Province. Education institutions and the industry must work together. This will allow software developers to share their practical experiences with learners. The need to establish localized forums where students and software developers share practical experiences is paramount.

5.2.1.10 Province of respondents

With reference to Table 4.11, the majority 80.6 percent of respondents are coming from the Western Cape Province. This is because the unit of analysis for this study is software developers based in the Western Cape Province. However, this suggests a need of similar studies for other provinces in order to understand the status quo of skills of software developers in South Africa as whole. One can further expand this study to cover the entire African continent in order to establish where Africa stands in the field of software development.

5.2.2 Software development technologies

5.2.2.1 Software development environment (IDE) tools

With reference to Table 4.12, the majority 52.9 percent of respondents use Microsoft Visual Studio as a software development tool. This indicates that, despite the fact that a company could have several IDEs, Microsoft Visual Studio is an integral part of software development activities for many software companies. Mastering Microsoft Visual Studio is critical for individuals looking for software development employment in the Western Cape Province. As it stands, it is clear that Microsoft Visual Studio is a basic tool that most of software developers utilize to perform their tasks. Institutions have a role to ensure that learners are capable of utilizing Microsoft Visual Studio as the main software development tool. Other important IDEs are Net Beans, Eclipse and Dreamweaver used by 49.5 percent (18 plus 16.5 plus 15) of respondents.

5.2.2.2 Programming technologies

With reference to Table 4.13, most of respondents are involved in developing web applications. Thus knowledge of programming technologies for web applications such as HTML, Java script, jQuery, CSS, XML, C#, PHP, ASP.NET, Ajax and Java is critical among software developers in the Western Cape Province. HTML, Java script, jQuery and CSS are critical client side technologies while C#, PHP and Java are critical server side technologies. The above-mentioned technologies should form the foundation of software development studies. The industry should regularly provide institutions with information regarding programming technologies required. This will ensure that institutions provide training that prepares learners to work as competent software developers in real life software projects.

5.2.2.3 Database Management System (DBMS)

With reference to Table 4.14, Microsoft SQL server is the database management system used by the majority 53.9 percent of respondents followed by MySQL used by 48.1 percent of respondents. Therefore, it is clear that Microsoft SQL server and MySQL should form the foundation knowledge of DBMS taught by institutions. Another important database management system is Oracle, which is used by 22.3 percent of respondents.

5.2.2.4 Code version control tools

As the software company grows, the number of clients increase and the number of software projects increase hence the need to have some means of storing, managing and controlling source codes. With reference to Table 4.15, the majority 35.9 percent of respondents use Subversion as their preferred source code version control software. This indicates that knowledge of Subversion is paramount for software developers. Another important source code control tool is Visual studio team system used by 23.8 percent of respondents. Despite the fact that 23.8 percent do not use source code control tools, knowledge of using the tool remains important especially in large software projects. In addition, it was reported that Team Foundation Server (TFS) is being used for source code control, data collection, reporting, and project tracking and task allocation among project members in a given software project. TFS is thus important in big and collaborative software development projects.

5.2.2.5 Application servers

With reference to Table 4.16, Microsoft server products are used by 46.1 percent of respondents followed by Apache servers used by 42.7 percent. Thus, knowledge of Microsoft server products and Apache servers is paramount for software developers. Institutions should work together with the industry in order to ascertain important servers that should form the foundation of learners. Experienced software developers should visit institutions, share work experiences regarding application servers and possibly demonstrate to learners the role application servers play during the execution of real life software projects.

5.2.2.6 Software installation environment

Most of institutions focus on teaching software development for the windows environment. With reference to Table 4.17, 79.1 percent of respondents target windows as their deployment environment. However, 40.8 percent of respondents develop software systems for the Linux environment and 17.5 percent develops for Unix environment. With advancement of open source software, the proliferation of software systems based on open source is inevitable. Thus, it is important for institutions to realize that knowledge of both Windows environment and Linux environment is critical among software developers. Linux is particularly used for servers as the operating system.

5.2.3 Tasks performed by software developers

Some of respondents do specialize and perform specific software development tasks for specific phases of Software Development Life Cycle (SDLC). For instance, there are software developers who focus on analysis of user requirements, developers who focus on software testing, developers who focus on software deployment and those who focus on software maintenance and support. With reference to Table 4.18, 82.5 percent of respondents deal with the design of software solution, the majority 88.8 percent of respondents deal with coding of a software solution and 76.2 percent of respondents deal with test of software solution. Thus the top three tasks performed by software developers in descending order are:

- i. Coding of software solution*
- ii. Design of software solution*
- iii. Test software solution*

While there are specialization areas in software development, it is paramount for software developers to have a thorough and grounded knowledge of tasks performed during all phases of SDLC. SDLC is a compass to software developers because it provides directions to be followed during the execution of a software project. SDLC is the foundation of software development. While there are many text books on SDLC, institutions should work with the industry in order to convey the practical experience of real life project to learners and compliment knowledge gained from textbooks. As far as SDLC is concerned, software project managers and experienced software developers should be allowed to visit institutions and talk to learners on current real life software projects. This will help institutions understand to what extend learners need to know SDLC and tasks performed during SDLC. It is recommended that SDLC form the foundation of internship programmes for learners wanting to work as software developers.

5.3 Discussion on results of qualitative data analysis

As indicated in chapter four, qualitative data analysis was performed on open-ended questions of the survey questionnaires and interviews. This section discusses results of data collected via open-ended questions of questionnaires and interviews. This data reveals software development tools, tasks performed by software developers and skills required by software developers.

5.3.1 Skills of software developers (Open-ended question of questionnaires)

Software development is a more practical field than a theoretical field. Institutions should ensure that both theoretical principles and practical elements of software development are understood by learners. Students should know and be able to do. It is the practical elements that differentiate software development from other careers. Software developers should be capable of practically performing tasks such as writing codes. Hence there is a need for the industry to share work experiences with learners via specific skills development programmes.

A skills development programme such as internship can provide learners with essential practical experience to undertake future real life software projects. The theoretical understanding of skills is one and the practical understanding is another. It goes without saying that both theoretical approach and practical approach to teaching software development are important and complement each other. With reference to Table 4.19, Table 4.20, Table 4.21 and Table 4.22, critical skills of software developers are:

- i. *Technical skills: ability to apply technology in solving problems*
- ii. *Analytical and thinking skills: ability to understand and solve problems*
- iii. *Communication skills: ability to communicate messages correctly, accurately and appropriately*
- iv. *Research skills: ability to search for information in order to get solutions timeously*
- v. *Focus on customer needs: ability to bear customer needs in mind all the time while designing and developing software solutions*
- vi. *Innovative: ability to be creative and proactive while solving problems*
- vii. *Team work: ability to work within a team without compromising self-identity*
- viii. *Reuse of code: ability to use what is already available, whenever possible instead of inventing the wheel*
- ix. *Experience: ability to apply previous experience appropriately while solving business problems*
- x. *Management skills: ability to plan and work according to plan*

5.3.2 Discussion on interviews

With reference to data analysis of interviews in chapter four, software developers raised four critical points of interests as discussed in the following section.

5.3.2.1 Major skills of a software developer

All interviewees indicate that the most critical skill of any software developer is technical skills. This is because a software developer is a problem solver who uses software technologies to solve business problems. A software developer must have the ability to write codes in order to address business requirements. A software developer should know how to write codes to solve business problems. A software developer uses code to solve business problems.

As such until one is capable of writing codes in order to solve business problems, he/she is not a software developer. Thus if one wants to become a software developer, one has to learn and master the science or art of writing codes that solves business problems, which means mastering specific programming technologies. A software developer is a master of programming technologies. However, these programming technologies are of value if relevant to requirements of the industry and address the needs of the industry.

Based on the premise that a software developer has a responsibility of writing codes, a software developer is a technical person and a practical individual. Institutions should train learners to write codes instead of focusing on theory of software development. Institutions should train learners to become technical and practical individuals as opposed to individuals who know lots about theory but little on practical. Institutions should ensure that learners understand theories of software development but most importantly understand how to apply that theory at a practical level. The software industry view software developers as people who can do as opposed to individuals who understand theory. However in order to do one needs to understand what to do and how to do which is a challenge to be address by both institutions and the industry.

5.3.2.2 Technical ability of new software developers

Interviewees indicate that newly recruited software developers who mostly have tertiary qualifications are still unable to perform basic tasks of software developers. It was reported that new recruits have very limited practical knowledge on technologies used to develop software systems. They have to learn most of technologies from the scratch while in the industry. This creates a steep learning curve and is costly. It takes long time before new recruits become useful and productive in software

projects. This can be attributed to the fact that the industry uses software technologies that are either not taught at a required standard or not taught at all by tertiary institutions. This confirms a gap between technologies taught by institutions and technologies used by the industry. Institutions and the industry must work together in order to bridge this gap of knowledge. This will avoid institutions from teaching out-dated software technologies that are no longer used by the industry and focus on teaching technologies required by the industry.

5.3.2.3 Skills regarding software development for mobile devices

Business companies see opportunities in investing in software projects targeting mobile devices. Nevertheless, as indicated by some of interviewees, there is lack of software developers with experience in developing software for mobile devices. One company of the six companies interviewed, reported to use software developers from India and Ukraine since it is difficult to find local software developers with relevant technical experience for mobile devices.

This lack of software developers for mobile devices can be addressed if tertiary institutions provide courses specific for software development for mobile devices. Such course could focus on some of mainstream mobile technologies such as android, iPhone and windows 7 platforms. With the proliferation of mobile devices in the African continent, one envisages the need of technical skills to develop software for mobile devices. Hence the need for tertiary institutions to offer programmes specific to software development for mobile devices.

5.3.2.4 Software applications developed by most of companies

As indicated by data analysis of interviews in chapter four, development of web applications constitutes major part of software projects performed in the Western Cape Province. This is also confirmed by data on technologies used by software developers in chapter four where technologies for developing web applications constitute large percentage. This is an indication that demand for technical skills for developing web applications is high and hence technical knowledge of developing web applications is fundamental to software developers.

Therefore, it is recommended that institutions offer programmes that will enable learners acquire maximum technical knowledge of developing web applications.

While there are many Internet technologies to learn, institutions should identify web technologies used by most of business and software companies to be the core of software courses offered. For instance, chapter four reveals technologies used by most of software developers in the Western Cape Province. It thus makes lots of sense for institutions to offer programmes to address technologies used by the majority of software developers. Since software technology is dynamic, institutions should work with the industry in order to ensure that institutions offer relevant software technologies to address present business needs. This will ensure that education offered is always in sync with skills required.

5.4 Chapter Summary

This chapter discussed research data collected during the research and provided brief recommendations to both institutions and the industry. The discussion focused on the profile of software developers, software development tools, tasks performed by software developers and skills required by software developers.

5.5 Conclusion

Having discussed results of data analysis, it is clear that there is a wide gap between knowledge provided by institutions and that required by software developers to execute their software development tasks. Knowledge of technologies required by software developers, tasks performed by software developers and skills required by software developers is critical if the gap is to be bridged and consequently eliminated for the sake of success in software projects.

The following final chapter provides detailed recommendations going forward. This is in order to improve competencies of software developers and consequently address the shortage of skilled software developers.

CHAPTER SIX: RECOMMENDATIONS AND CONCLUSION

“If you give people tools, and they use their natural ability and their curiosity, they will develop things in ways that will surprise you very much beyond what you might have expected”
(Bill Gates)

6.1 Introduction

This final chapter presents research summary, recommendations and conclusion. Summary of this research is indicated in section 6.2 and SDCF (Software Developer Competency Framework) is revealed in section 6.3. Recommendations are discussed in section 6.4, followed by message to institutions and the industry in section 6.5. Limitations of the study and future studies are discussed in section 6.6 and section 6.7 respectively. Finally, the chapter ends with conclusion in section 6.8.

6.2 Research summary

The research was conducted in the Western Cape Province and the unit of analysis was software developers. The study employed both quantitative and qualitative methodologies in order to attain the benefit of methodology triangulation. As such, an online survey questionnaire was prepared and the survey link was distributed among software developers. Questions on the survey questionnaire aimed at collecting data pertaining to software tools used by software developers, tasks performed by software developers and skills required by software developers. The collected data were analysed using PASW Statistics 18 and Excel 2010. Moreover, in order to complement this study, interviews were performed. Both quantitative and qualitative data analysis were performed and the detailed result of data analysis is presented in chapter four and discussed in chapter five.

The objective of this study is to develop a competency framework that could be used by both institutions and the industry to improve competency of software developers, hence the name Software Developer Competency Framework (SDCF). SDCF brings about the realization of relevant competencies of software developers required by the software industry. As indicated in the recommendations section below, knowledge of SDCF is critical for the betterment and success of software projects. The developed SDCF is revealed in the following section (Figure 6.1) and explained thereafter.

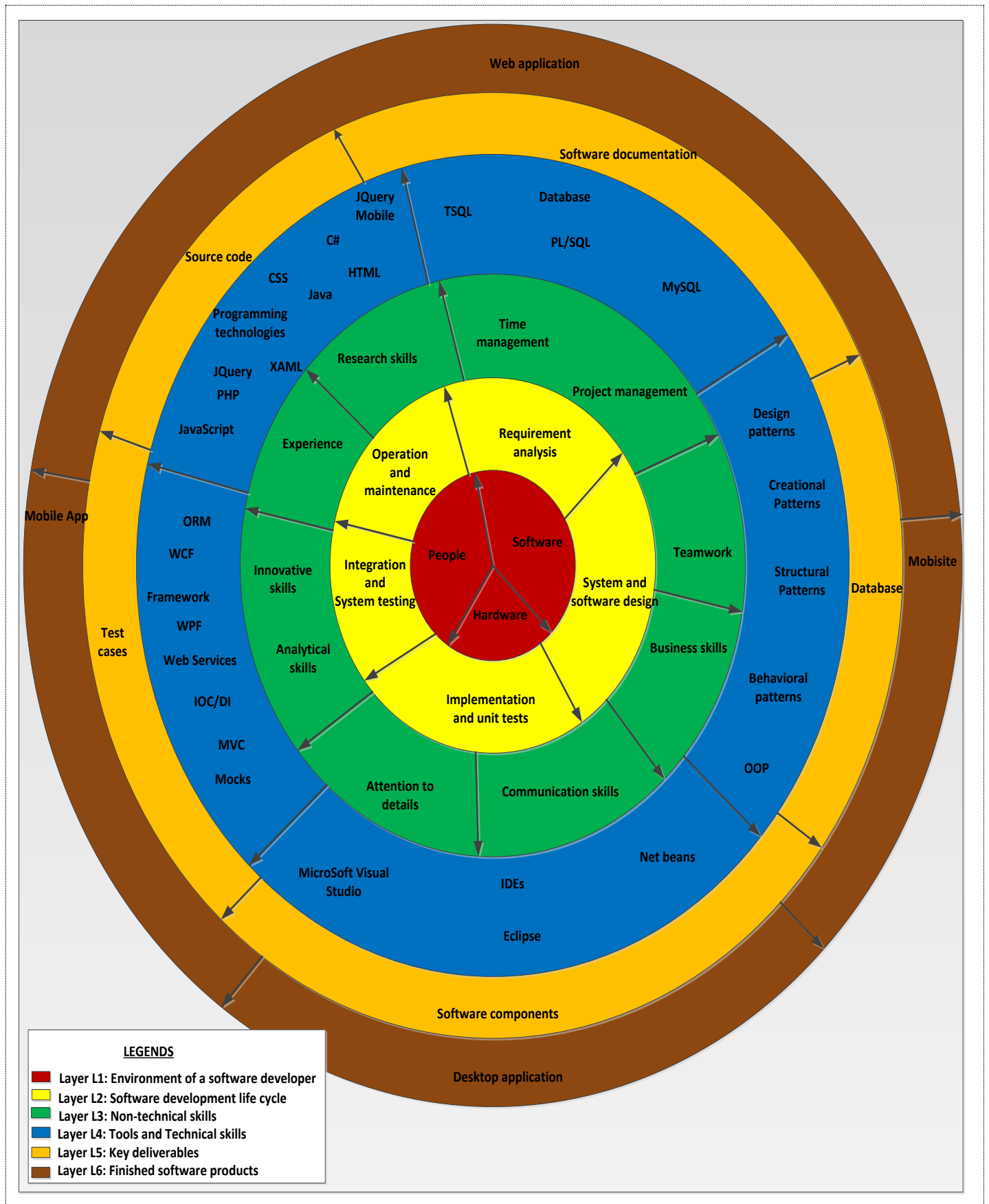


Figure 6.1: Software developer competency framework (SDCF)

Chapter Six: Recommendations and Conclusion

6.3 Recommended SDCF revealed and explained

Based on literature review, surveys, interviews and findings of both quantitative and qualitative data analysis performed during this study, Software Developer Competency Framework (SDCF) is hereby revealed and explained.

Data collected from intensive literature review on software projects, data collected via survey questionnaires and data collected during interviews underwent analysis. The comprehensive data analysis manifested identifiable patterns that resulted into the formation of Software Developer Competency Framework (SDCF), illustrated in Figure 6.1 above. SDCF encompasses knowledge based on empirical and factual data. Competencies required by software developers to develop software systems can easily be identified by carefully investigating components or layers of SDCF. SDCF is the essence and purpose of this research. Because it is composed of tasks, tools and skills of software developers, SDCF is a cornerstone of success in software development projects. It is hereby advised and recommended that SDCF form the foundation of the curriculum for all software development studies offered by tertiary institutions.

With reference to Figure 6.1 above, SDCF is composed of six layers (L1 to L6) as indicated below.

- i. Layer L1: Environment of a software developer*
- ii. Layer L2: Software development life cycle*
- iii. Layer L3: Non-technical skills*
- iv. Layer L4: Tools and technical skills*
- v. Layer L5: Key deliverables*
- vi. Layer L6: Finished software products*

Competencies required by software developers are rooted within layers of SDCF. Each layer is composed of logical partitions or units to represent the must-know elements of software developers. In addition, each layer is indicated with unique colour for identification purposes. Layer L1, with red colour, is the innermost layer while layer L6, with brown colour, is the outermost layer. Knowledge embodied in SDCF grows outwards from layer L1 towards layer L6 hence indicated by arrows pointing outwards. While some layers can be interchanged, the inner layer forms the foundation of the next outer layer.

Chapter Six: Recommendations and Conclusion

The following section describes all six layers in details.

6.3.1 Layer L1 with red colour: Environment of a software developer

The environment of a software developer is composed of three main components; software, hardware and people. The extent to which one component affects a software developer depends on the nature, complexity and size of a software project. Predominantly the software component takes big portion of this layer. However following the user-centred approach to software development where users are involved from the beginning to the end of a given software project, the people component is critical. This is because in big software projects, usually there is continuous communication and feedback between software developers, management and users. During this study, some of interviewees indicated that when working on big software projects, they have continuous interactions with other software developers, management and users, hence the people component.

Regarding hardware, software runs on hardware hence software developers are exposed to different kind of hardware such as computers, mobile devices and many other devices.

Recognizing these components prepares individuals to understand their environment in which they will spend most of their time as software developers. It goes without saying that an individual with passion in software, hardware and people can make a great software developer.

6.3.2 Layer L2 with yellow colour: Software development life cycle

The software development life cycle (SDLC) used in a given software project depends on the nature, complexity and size of a software project. There are primitive and ancient traditional methodologies such as waterfall, typically used in small projects where user requirements are clearly defined and understood. In this case, a project is divided into distinct phases where one phase is completely done and closed before proceeding with the next phase. Then there are agile methodologies that involve incremental and iterative approaches. Agile methodologies are preferred and largely implemented in big software projects. The reasoning behind agile approaches is based on the premise of discovering risks, uncertainties, understand

Chapter Six: Recommendations and Conclusion

the complexity of user requirements and address dynamic changes in user requirements. In agile methodologies, software systems are developed in continuous iterative manner where each iteration results into the release of functional software components deployed for clients to use immediately. During this study interviewees indicated to prefer agile methodologies. Some of interviewees mentioned that agile methodologies cater for the “learn as you go” mentality to allow discovery of unknown during the software development process.

As indicated in chapter four (Figure 4.18), during the survey, when asked to indicate categories of their tasks as software developers, more than half of respondents indicated to perform all categories of tasks in the software development life cycle. In spite of differences between primitive and agile methodologies, as indicated by layer L2, any software development life cycle has five main phases as described below:

- i. *Requirement analysis:*
During this phase, the objective is to understand the problem in order to solve that problem.

- ii. *System and software design:*
In this phase, feasible solution to address the problem is designed.

- iii. *Implementation and unit tests:*
In this phase, software components are developed and unit tested to ensure that software components perform as expected.

- iv. *Integration and system testing*
Software component are integrated into the system and the entire system is tested to ensure that user requirements are met.

- v. *Operation and Maintenance:*
Software solution is deployed to clients and maintenance continues to address problems or bugs and streamline software features.

All tasks performed by software developers falls into the above-mentioned phases and are critical to the success of any software project. An aspiring software developer should aim at understanding the theory and practical behind the above-

Chapter Six: Recommendations and Conclusion

mentioned components of Layer L2. As indicated in survey and interviews conducted during this study, most of software companies require that software developers be fully versed in SDLC. SDLC translates into tasks performed by software developers. Tasks refer to software development activities performed by software developers. Therefore this layer L2 is a must know to all software developers.

6.3.3 Layer L3 with green colour: Non-technical skills

Because people constitute a significant portion of Layer L1, software developers need skills that will enhance their tasks indicated in Layer L2. These skills tend to be applicable to most of careers where people are involved and where there is a constant need to understanding new insights of a given field. These skills are sometimes known as soft-skills, people skills or non-technical skills. These skills refer to attributes that enhance job performance of software developers and determine how well a software developer executes software development tasks.

In this study, these skills are referred to as non-technical skills as they are not technology specific. According to survey and interviews conducted for this study, critical non-technical skills are teamwork, communication skills, analytical or thinking skills, attention to details, research skills and project management skills.

As indicate in chapter four (Figure 4.19), the survey reveals that top three non-technical skills are:

- i. *Analytical and thinking skills; agreed by 96.6 percent of respondents*
- ii. *Communication skills; agreed by 94.7 percent of respondents*
- iii. *Ability to do self-study and research; agreed by 93.7 percent of respondents*

6.3.4 Layer L4 with blue colour: Tools and technical skills

As indicated in Layer L1, software, hardware and people form significant components of the environment of software developers. As such in order to execute tasks indicated in Layer L2 software developers require skills to deal with software and hardware. These skills are technology dependent as they require the understanding and ability to apply technology using specific tools to perform specific tasks of Layer L2. Tools refer to any software that a software developer utilize in order to develop other software systems. The skills of using tools are known as technical skills hence

Chapter Six: Recommendations and Conclusion

technical skills refers to know-how of using software tools to perform software development tasks.

This layer of technical skills is dynamic and hence demands software developers to keep on updating their technical skills. This layer reflects relevant software technologies used by the industry. This layer L4 must be carefully observed by institutions and continuously updated by the software industry in order to prevent discrepancies of knowledge provide by institutions and that required by the industry.

According to this study technical skills demand proficient in understanding and mastering of IDEs, programming technologies, databases, software frameworks, design patterns and OOP concepts. Research skills play critical part in understanding technical skills required by software developers. According to survey and interviews performed this layer L4 is considered as the most important and critical determinant of the software development career. All software companies initially employ software developers based on their technical ability indicated by Layer L4.

6.3.5 Layer L5 with orange colour: Key deliverables

Using non-technical skills and technical skills indicated in Layer L3 and Layer L4 respectively, in order to perform tasks indicated in Layer L2, software developers have key deliverables to accomplish. According to survey and interviews performed in this study, key deliverables of individual software developers are source code, working software components, test cases and documentation. It is important that this layer be recognized by software developers because this layer L5 determines what deliverables are expected when tasks are assigned to software developers. It is worth knowing that when one is aware of key deliverables expected, one can aim at achieving delivery of the required deliverables.

6.3.6 Layer L6 with brown colour: Finished software products

The final layer of SDCF is Layer L6 which constitutes of finished software products developed by software developers in a given software project. According to this study, final products of software projects are web applications, desktop applications and mobile applications. With reference to data analysis of survey and interviews conducted during this study, web applications constitute a significant portion in the finished software products developed in the Western Cape Province. This layer L6

Chapter Six: Recommendations and Conclusion

clearly communicates a message that skills related to development of web applications is in demand in the Western Cape Province. As such to avoid shortage of skilled software developers for web applications, institutions must produce more software developers who can thoroughly develop web applications. Moreover, with reference to interviews performed, knowing that already there is lack of skilled software developers for mobile applications, institutions must establish software development courses related to development of software for mobile devices in order to address lack of skilled software developers for mobile devices.

6.4 Recommendations

Based on merits attributed to SDCF, the researcher hereby recommends and submits that SDCF be adopted as a foundation for curriculum of software development courses offered by all tertiary institutions.

SDCF encompasses both technical and non-technical elements required by an individual to function as a competent software developer. This framework is an attempt to bridge the gap between institutions and the industry such that software programmes offered by institutions are relevant to industry needs. Many merits can be accomplished if SDCF is used as a foundation of policy document for software development programmes offered by institutions. Likewise, there will be many demerits haunting the software development industry, if SDCF is ignored and not used as a base of software development studies. The following section outlines possible merits that can be triggered by the adoption of SDCF to address competencies of software developers.

6.4.1 Bridge of tertiary institutions and the industry

Primarily SDCF is a bridge that links institutions and the industry. By focusing on components of SDCF, institutions will be able to offer relevant software studies to address skills shortage of software developers. In addition, software developers produced by institutions will have quality knowledge of both technical and non-technical skills essential to work as competent software developers. It is worth to mention that institutions will be able to offer software programmes relevant to the industry requirements. Unless this gap is bridged the degree of discrepancy of knowledge of software studies offered by institutions and relevant software skills

Chapter Six: Recommendations and Conclusion

required by the industry will continue to widen and definitely affect the software industry and businesses at large. The more the gap increases the more the shortage of skilled software developers increases. The SDCF developed during this study has the potential of bridging the knowledge gap between institutions and the industry as far as software development is concerned.

6.4.2 Smooth learning curve to newly recruited software developers

This study acknowledges that software development field is a dynamic environment and requires software developers to continue learning new technologies while working. Nevertheless, learning curve among new software developers can be smoothed and shortened if institutions offer courses that are relevant and in sync with requirements of the industry. This study reveals that there are software developers who had to resign their job because they could not cope with the steep learning curve to learn technologies in order to work as software developers. Also it was found that, despite having tertiary qualifications, most of new recruited software developers are considered as negative resources to companies. This is because new software developers have to undergo lengthy and intense training in order to bring their skills to a level that they could perform tasks and become useful in software projects. The lengthy and steep learning curve is costly to companies because it takes long time before individuals become productive. For that reason, if implemented, SDCF will reduce the amount of learning required after employment because new software developers will be already exposed to the relevant technologies used by the industry.

6.4.3 Employable software developers

Following the need of businesses to explore and invest in the application of information technology in order to attain maximum profit, software developers continue to be a scarce resource. Therefore an individual with relevant competencies as a software developer has a greater chance of being employed by business organizations and software companies. Such an individual has knowledge of most of relevant technologies used by the industry and is useful in software projects almost immediately. Such a software developer is definitely employable. Long steep learning curve and costs incurred before an employee becomes productive to the company are reduced and kept to a minimum level, if the employee is technically

Chapter Six: Recommendations and Conclusion

knowledgeable. The guaranteed way of getting a job as a software developer and making a software developer easily employable is to master technologies used for software development in a given industry. This is only possible if SDCF is used as a point of reference for software development studies offered by all institutions. SDCF will make a person with tertiary qualification in software development to be in demand and market-sought both locally and internationally.

6.4.4 Production of software development experts

It is expected that institutions are places to learn and acquire knowledge that can be applied in the industry. Likewise, one expects that the industry is a place to apply acquired knowledge in order to execute tasks. Nevertheless, according to this study, most of software developers learn relevant technologies once employed in the industry. This is because knowledge acquired from institutions does not suffice and misses lots of critical ingredients required by software developers to confront real life software projects. Using SDCF as a point of reference and benchmark will produce software developers who are experts in their field of software development. This will also encourage employees to come back to institutions in order to upgrade and update their software development skills. Institutions will be places to learn and the industry will be a place to work and apply acquired knowledge from institutions.

6.4.5 Reputation of institutions regarding software programmes

Following the fact that SDCF encompasses relevant skills, institutions offering relevant skills required by the industry will have competitive advantage over other institutions. This will increase the reputation of tertiary institutions towards acceptance by the industry. It is in the best interest of institutions to have a good reputation towards academic excellence. This could be achieved by using SDCF as a standard for software development programmes offered by tertiary institutions.

6.4.6 Alleviation of unemployment in communities

The research reveals that some of companies opt to outsourcing software developers from countries such as India and Ukraine. This was predominantly in skills pertaining to software development for mobile devices. SDCF depicts relevant skills required by the industry. Therefore if SDCF is used as a lens or compass to guide software development courses, competent software developers will be produced locally and

Chapter Six: Recommendations and Conclusion

the need to outsource skills will diminish. This will make local people more employable as opposed to relying on foreign supply of software developers. In the long run this will boost up employment and reduce unemployment rate affecting tertiary education graduates. Knowing that the alleviation of unemployment has positive effects to the development of our communities, adoption of SDCF is paramount.

6.4.7 Confidence in software programmes offered by institutions

Institutions will become centres of excellence in software development studies where advanced technical and practical skills are nurtured and acquired. Often people from the industry will be coming back to institutions in order to acquire practical skills on solving business problems using recent and advanced software technologies. This is possible if and only if institutions offer programmes that are relevant to address current industry needs. These are programmes that add great value immediately to the industry. SDCF will encourage the industry to utilize institutions in order to achieve that unique knowledge with value; knowledge embodied in SDCF and conveyed via software development programmes offered by institutions. It will make the industry to have confidence in software programmes offered by institutions. More can be achieved if SDCF is used as a standard for software development courses offered by tertiary institutions.

6.4.8 Technical and practical value on software programmes offered

Traditionally institutions are renowned for providing academic knowledge via teaching materials primarily based on text books. The benchmark of success is then gauged based on the extent to which an individual understood material learned from text books. The traditional means of benchmarking are examinations and tests where individuals are asked questions based on materials learned from those text books. As far as software development is concerned, this approach is limited and hence requires to be complimented by a practical approach to learning. It is the technical and practical elements that make software developers experts in the field of software development. Experts in software development are individuals who can think out of the box and take software technology to the extreme in order to address business problems. These are individuals who have solid technical ability to use software technologies to address business problems for the betterment of our communities at

Chapter Six: Recommendations and Conclusion

large. Individuals of such a technical calibre, no sooner than they graduate from tertiary institutions, they will be employed or self-employed almost immediately. This is because there are lots of gaps particularly in Africa following the big digital divide as compared to the developed world. When an individual is equipped with technical and practical knowledge, that person has the power of thinking out of the box; the thinking is not limited to shallow knowledge based on text books only. As indicated in Franco (2012), software development is a highly technical and specific discipline. The advanced level of mastering technical skills can be achieved via implementing SDCF to address relevant technical skills required by software developers.

6.4.9 Poverty alleviation in communities

Poverty in most of African communities is an indication of opportunities not yet realized. These opportunities can be exploited using software technologies. Basic things like bringing community services close to community members can be achieved via software technologies. Nevertheless, it requires solid practical and technical knowledge of software development to be able to perform software projects that can alleviate the suffering of poor communities. Having good ideas is one thing but putting those ideas into action is another. It is important to recognise that action speaks louder. However actions can be executed if one has technical and practical ability to undertake the execution of ideas. As far as software development is concerned, one must have technical ability to bring about changes in our impoverished communities via software technologies. It is not uncommon for tertiary education graduates to be unemployed. Nevertheless, it does not make sense when an individual graduates from tertiary institutions with a major in software development studies and remain unemployed despite all opportunities in our impoverished communities that can be addressed by developing software systems. SDCF can ensure that software developers have knowledge to address current business needs and community problems. SDCF can play a role in alleviation of poverty in our communities.

6.4.10 Centre of technical and practical excellence for software studies

More can be achieved if SDCF is used as a standard for software development courses offered by tertiary institutions. As already mentioned, it is expected that institutions are places to learn while the industry is a place to do and apply

Chapter Six: Recommendations and Conclusion

knowledge acquired from institutions. Nevertheless, this research indicates that most of software development graduates are not capable of performing their tasks as expected as such they have to learn most of technologies from the scratch while in the industry. This must change so that institutions become centres for learning and acquiring technical and practical excellence. This is especially important to software development studies where graduating from tertiary institutions should, on its own, be an indication that a graduate has a solid technical background on software technologies relevant to the industry needs. Graduates should be able to demonstrate practical and technical ability of applying advanced software technologies to address business challenges and community problems. This can be achieved via offering software courses based on SDCF.

Unless institutions provided relevant skills for software development, institutions will cease to be useful in bringing software development skills to addressing business challenges facing the industry and communities at large. SDCF is geared towards ensuring that institutions offer programmes that are relevant to address problems in our communities in terms of providing technical, practical and sustainable solutions. SDCF will add value to the quality of education required by software developers to address business needs and confront challenges affecting our communities. SDCF has the potential of revolutionizing the status quo of unemployment rate and brings about positive impact to our impoverished communities. SDCF will bring the industry and institutions together. As a result institutions will become centres of technical and practical excellence where individuals from the industry will be coming back to institutions in order to upgrade their software development knowledge.

6.5 Message to institutions and the industry

While SDCF is a tool that has the potential to revolutionize the software industry for better, it does require a conducive and supportive environment in order to flourish. Both institutions and the industry have key roles to play to make SDCF a reality in the realm of software development. This section thus provides advice on how to make SDCF a reality for the betterment of the software industry and business as a whole.

There is direct dependency between SDCF and business challenges which are dynamic in nature. In order to achieve the primary goal of SDCF which is to ensure

Chapter Six: Recommendations and Conclusion

maximum level of competency among software developers, information on the competency framework must be consistent, correct and relevant to industry needs.

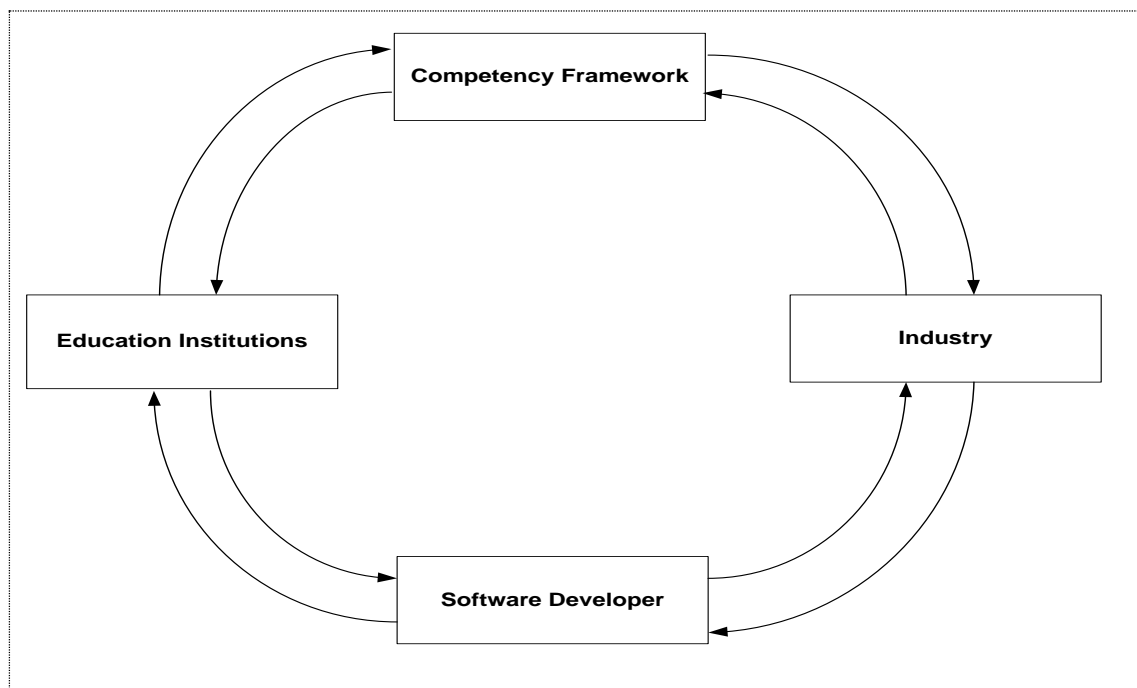


Figure 6.2: Competency Cycle

Consistency, correctness and relevancy of SDCF can be achieved if a strong communication link among institutions and the industry is established, strengthened and maintained. This will allow smooth back and forth flow of information among institutions and the industry. With reference to figure 6.2, SDCF should be executed in a two way round robin fashion referred to as “competency cycle” where both institutions and the industry take responsibility in updating SDCF. This will ensure that SDCF is up-to-date and hence consistent, correct and relevant to industry requirements.

The primary goal of any institution is to inculcate knowledge of a given discipline to learners. Nevertheless, knowledge becomes wisdom if it is consistent, correct and relevant to circumstances; hence can be applied to solve problems. It is acknowledged that the content of SDCF will continue to change as information technology advances. Therefore, institutions and the industry should ensure that SDCF has updated and researched information. Then Institutions should process the SDCF and use it as a basis for curriculum offered to learners who are the future software developers. SDCF is a critical tool for the success of software projects in the

Chapter Six: Recommendations and Conclusion

Western Cape Province. This is because SDCF will ensure that learners have solid foundation of relevant skills to become competent software developers.

The researcher further proposes the need for every software company or business organization to have an identifiable contact person whose main job includes liaising and regularly communicating with institutions all information regarding the industry needs as far as SDCF is concerned. On the other hand institutions need to have an office whose main task will be dedicated to processing information received from the industry in order to update SDCF and share information with lecturers. This will ensure that the industry needs are in sync with education offered by institutions and hence relevancy, consistency and correctness of skills offer via SDCF.

It was found that many software companies and business organizations do not participate in internship programs consequently many students are not given opportunity for exposure to real life software projects in the industry. While traditionally it has been the discretion of the companies to participate in the internship programs, it is hereby proposed the need to make participation in internship programmes compulsory to all software companies and business organizations in the Western Cape Province. By so doing all students undertaking software development studies will have the opportunity for exposure to real life software projects. This will enhance their skills as competent software developers.

The participation of the community members has potential to change the community at large. Minani and Parker (2009:92) report about a community initiative group known as “The Impact Team” that plays key roles in combating drugs and gangsterism in the Cape Flats area, in the Western Cape Province. Similar community initiatives that focus particularly on software projects could assist in bringing the software development community together. This can create a platform where experience software developers, novices and learners share software development technologies, knowledge and experiences. This kind of community participation will greatly improve and enrich the body of knowledge pertaining to critical competencies of software developers in the Western Cape Province. As such, the researcher hereby challenges the industry and education institutions to pioneer community initiative groups focusing on software development initiatives.

Chapter Six: Recommendations and Conclusion

Despite of all merits attributed with the use of SDCF in institutions, it is important to be mindful of the fact that there are stakeholders, particular lecturers, who may not reciprocate their feelings towards supporting the use of SDCF as a standard for software development courses offered by institutions. This is because SDCF will change the way to deliver software development studies. The use of SDCF may translate to more workload to lecturers in order to teach skills that may not be familiar to lecturers. SDCF will require some of stakeholders, for instance, to research and continue learning in order to be updated with current and relevant software development technologies in the market. This is opposed to the traditional way of teaching using textbooks only where same textbook is taught after every year resulting into delivery of irrelevant and inconsistent software development skills. Teaching same text book over and over again is definitely very easy than teaching new and relevant skills every time. Hence, it is to be anticipated that some of stakeholders, particularly lecturers, may not support the use of SDCF. However, following many merits associated with utilizing SDCF as a point of reference for software development studies, it is hereby submitted and recommended that this research be used as a foundation for the policy document used by institutions for software development studies. No sooner than SDCF is adopted as foundation for software development studies, institutions will be obliged to offer software development programmes that are relevant to the industry requirements and address community challenges. SDCF will greatly address shortage of competent software developers in the South Africa and Africa as a whole.

Lastly but not the least, the researcher submit that the industry be made part and parcel of institutions were experienced software developers from the industry regularly visit tertiary institutions to speak to students, share knowledge and experiences about the nature of real life software projects as per SDCF. Not only will this motivated students to participate in software development projects but it will create an impact on the quality and standard of future software developers. Eventually, this will reduce initial training costs incurred by most of software companies and business organizations in terms of training and time.

6.6 Limitations of the study

Although many software companies and business organizations participated in this research, there is poor cooperation and weak relationships between the industry and

Chapter Six: Recommendations and Conclusion

education institutions. Various companies partake in internship programmes while other companies do not participate in internship programmes at all. During this study it was found that the wide communication gap among institutions and the industry is the main limiting factor. This communication gap affects the flow of information among institutions, software companies and business organizations. Some of companies, for instance, did not participate in this research because they have no formal relationships with institutions. As such there is limited awareness regarding roles that institutions and the industry could play together for the betterment of the software development industry in the Western Cape Province. Cooperation and relationships among institutions and the industry can be strengthened by dissemination of information regarding roles that institutions and the industry can play in order to improve competencies of software developers. Dissemination of information will promote awareness of the importance of SDCF among institutions, software companies and business organizations.

6.7 Future studies

The focal point for this research was the Western Cape Province. As such, other provinces were beyond the scope of this study. Nevertheless, the researcher proposes that similar studies be undertaken in other provinces of South Africa. In addition, a comparative study among countries regarded as top and advanced in the field of software development should be performed. For instance, India, Japan, Finland and USA are regarded as being advanced as far as software technology is concerned. It will be beneficial to understand and learn from masters and powerhouses of information technology.

6.8 Conclusion

Soren Kierkegaard, The Danish philosopher once said, "Life can only be understood backwards but it must be lived forwards" (Nosotro, 2003). SDCF is a product of a research based on the past and present status quo of the software development projects. Information from an intensive literature review on software development projects and analysis of data collected via interviews and survey questionnaires resulted into the formation of SDCF. SDCF encompasses knowledge based on empirical and factual data. This conforms to Arthur Conan Doyle, The British philosopher who once said, "It is a capital mistake to theorize before one has data" (Doyle, 1887). Based on the evidence that most of work performed by software

Chapter Six: Recommendations and Conclusion

developers is maintenance of software systems, software developers should write software codes that are easily understandable by other software developers. SDCF equips software developers with knowledge to writing readable and maintainable software codes. This echoes the famous saying by Martin Fowler, author and international speaker on software development, who once said, “Any fool can write code that a computer can understand. Good programmers write code that human can understand” (Fowler, 1999:17). Similarly, Martin Golding, the computer scientist, once said, “always code as if the guy who ends up maintaining your code will be a violent psychopath who knows where you live” (Mathew, 2012). SDCF equips software developer with skills to write maintainable software codes.

In addition, debugging of software goes hand in hand with maintenance work. Since debugging is the process of removing bugs from a software system, one expects that writing software codes should not be the process of introducing bugs to software systems. However based on current status quo of software projects, programming is like a process of putting bugs into software systems, which resonates with the late Edsger Wybe Dijkstra, the computer scientist, who once said “if debugging is the process of removing bugs, then programming must be the process of putting them in” (Abbot, 2003:113). SDCF intends to change this mind-set so that the introduction of bugs into software system is minimal and hence minimize the debugging process of software codes. As highlighted by Dijkstra (1972), effective programmers should not waste time with debugging; they should not introduce bugs in the first place. SDCF will equip software developers with skills required to write bug-free software codes.

Most importantly, SDCF is a cornerstone of software development. SDCF is a tool and a solid foundation of competent software developers. If utilized well, SDCF can equip software developers with knowledge to master the art or science of software development. The World Bank (2008) has it that innovations have tremendous positive impact towards the economic growth of a given country. And SDCF is a catalyst capable to revolutionize technological innovations in the software industry. Coupled with their natural ability and curiosity, software developers will be able to write quality software systems, bugs will be minimal and success in software development projects will improve tremendously. This agrees with the idea of Bill Gates, The Co-founder of Microsoft (the largest software company in the world), who once said “If you give people tools, and they use their natural ability and their

Chapter Six: Recommendations and Conclusion

curiosity, they will develop things in ways that will surprise you very much beyond what you might have expected” (Kory, Chasser & Fingerhut, 2006). If applied appropriately, the adoption of SDCF by both institutions and the industry has the potential to greatly improve knowledge, skills, quality and standard of competencies of software developers. It goes without saying that knowledge of SDCF is critical among both institutions and the industry. Based on the above-mentioned merits of SDCF, the researcher submits that SDCF be adopted as a foundation of curriculum of software development programmes offered by tertiary institutions.

In conclusion, SDCF is paramount in ensuring that software developers are competent enough to deal with dynamic business challenges. SDCF can ensure that education offered by institutions is relevant to industry needs. Institutions will produce competent software developers if education offered is consistent and relevant to business requirements. Software companies and business organizations stand a big chance to increase their return on investment (ROI), if competencies of software developers are improved. This can be achieved by the use of SDCF which enforces the must know elements of software developers. SDCF is a depository of software development knowledge. SDCF is a central point that link institutions and the industry together. While institutions can provide solid knowledge about software development, only the industry can provide that practical experience of real life software projects. Therefore, in order to ensure maximum competency in software development, industry and institutions must work together towards adoption of SDCF.

LIST OF REFERENCES

- Abbott, D. 2003. *Linux for Embedded and Real-time Applications*. 1st ed. Oxford: Elsevier Science
- Agarwal, N & Rathod, U. 2006. Defining 'success' for software projects: An exploratory revelation. *International Journal of Project Management*, 24(2006):358-370.
- Ahn, Y. S., & McLean, G. N. 2008. Competencies for port and logistics personnel: An application of regional human resource development. *Asia Pacific Education Review*, 9(4): 542 – 551.
- Andrew, S., Salamonson, Y. & Halcomb, E.J. 2008. Integrating Mixed Methods Data Analysis: An Example Examining Attrition and Persistence of Nursing Students. *International Journal of Multiple Research Approaches*, 2: 36-43.
- Ardalan, K. 2010. Globalization and Global Governance: Four Paradigmatic Views. *American Review of Political Economy*, 8(1): 6 - 43.
- Babbie, E. & Mouton, J. 2001. *The practice of social research – South African edition*, Cape Town: Oxford University.
- Beck, K. 2005. *Extreme Programming Explained*. London: Addison-Wesley.
- Bennett, K., Layzell, P., Budgen, D., Brereton, P., Macaulay, L. & Munro, M. 2000. Service-Based Software: The Future for Flexible Software. *Seventh Asia-Pacific Software Engineering Conference APSEC'00*: 214 – 221
- Bennett, S., McRobb, S. & Farmer, R. 2006. *Object-Oriented Systems Analysis And Design*. 3rd ed. Berkshire: McGraw-Hill.
- Berelson, B.1954. *Content analysis in communication research*. Glencoe III: Free Press.
- Bertolino, A. 2007. Software testing research: Achievements, challenges, dreams. *Future of software engineering*, IEEE Computer society:85-103
- Best, J. & Khan, J. 1989. *Research in Education*, Englewood Cliffs. New Jersey: Pearson Prentice Hall.
- Beznosov, K., Flinn, D. J., Kawamoto, S & Hartman, B. 2005. Introduction to Web services and their security. *Information Security Technical Report*, 10(1): 2 – 14
- Bhattacharjee, A. 2012. *Social Science Research:Principles, Methods, and Practices*. 2nd ed. Florida: University of South Florida, ISBN-13: 978-1475146127 and ISBN-10: 1475146124
- Boehm, B. 1983. Seven basic principles of software engineering. *The Journal of Systems and Software*, 3(1):3-24.
- Bornman, E. 2006. National symbols and nation-building in the post-apartheid South Africa. *International Journal of Intercultural relations*, 30(2006): 383-399.

APPENDICES

- Bosch, J. & Molin, P. 1999. Software Architecture Design: Evaluation and Transformation. In Proceedings of IEEE Engineering of Computer Based Systems Symposium, IEEE Computer Society Press, Los Alamitos, CA, pp. 4-10.
- Bosch, T. E. (2009). Using online social networking for teaching and learning: Facebook use at the University of Cape Town. *Communicatio: South African Journal for Communication Theory and Research*, 35(2), 185-200.
- Bowen, G. A. 2005. Preparing a qualitative research-based dissertation: Lessons learned. *The Qualitative Report*, 10(2):208-222.
- Brandt, I.G. 2006. Models of Internet connectivity for secondary schools in the Grahamstown circuit. Masters Thesis, Rhodes University. Grahamstown. January 2006.
- Brooks, F. P. 1987. No silver bullet: Essence and accidents of software engineering. *IEEE Computer*, 20(4):10-19.
- Burns, R. B. & Burns, R. A. 2008. *Business Research Methods and Statistics Using SPSS*. London: SAGE Publications.
- Burrell, G. and Morgan, G. 1979. *Sociological Paradigms and Organizational Analysis*. London, and Exeter, NH: Heinemann.
- Calitz, A. 2011. Averting an ICT Crisis
<http://mq.co.za/article/2011-07-11-averting-an-ict-crisis-101> [08 October 2011]
- Caldo, F. 2008. Skills Shortage in South Africa
<http://www.solidarityresearch.co.za/wp-content/uploads/2010/07/16-Skills-Shortage-in-South-Africa-Summary-facts-FJC-ET.pdf> [22 April 2010]
- Charmaz, K. 2006. *Constructing Grounded Theory: A practical guide through qualitative analysis*. London: Sage.
- Cloete, F. 2011. Seven: e-GOVERNMENT LESSONS FROM SOUTH AFRICA 2001–2011: INSTITUTIONS, STATE OF PROGRESS AND MEASUREMENT. *The African Journal of Information and Communication*, 2001. 1- 170.
- Colomo-Palacios, R., Casado-Lumbreras, C., Soto-Acosta, P., García-Peñalvo, F. J., & Tovar-Caro, E. 2013. Competence gaps in software personnel: A multi-organizational study. *Computers in Human Behavior*, 29(2), 456-461.
- Copeland, J. & Haemer, J.S. 2000. The art of software testing. *Server workstation expert*, 11(8): 42-45.
- Crowley, C., Harré, R. & Tagg, C. 2002. Qualitative research and computing: Methodological issues and practices in using QSR NVivo and NUD*IST. *International Journal of Social Research Methodology* 5(3):193-197.
- Curbera, F., Nagy, W. A & Weerawarana, S. 2001. Web services: Why and how
<http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.24.4565&rep=rep1&type=pdf>
[18 August 2008]
- De Vos, A., Strydom, H., Fouche, C. B. & Delport, C. S. L. 2002. *Research at Grass Roots*. 2nd ed. Pretoria: Van Schaik Publishers.

APPENDICES

Deitel, H. M. & Deitel, P. J. 2006. Visual C# 2005: How to Program. 2nd ed. New Jersey: Pearson Prentice Hall

Di Gregorio, S. 2000. Using Nvivo for your literature review. Strategies in Qualitative Research: Issues and Results from Analysis using QSR Nvivo and Nud*ist, London, 29-30 September, Institute of Education.

Dijkstra, E. W. 1972. The Humble Programmer.
<http://www.cs.utexas.edu/~EWD/transcriptions/EWD03xx/EWD340.html> [16 March 2012].

Dorfman, M. 1999. Requirements Engineering, SEI Interactive
<http://www.dimap.ufrn.br/~jair/ES/reqengback.pdf> [20 June 2009].

Doyle, A. C. 1887. A Study In Scarlet.
http://www.gutenberg.org/files/244/244-h/244-h.htm#link2H_4_0001 [07 May 2010]

Fowler, F.J. 2009. Survey Research Methods. 4th ed. USA: Sage Publications

Fowler, M. 1999. Refactoring: Improving the Design of Existing Code. London: Addison-Wesley

Franco, S. 2012. Process key to successful software development
http://www.itweb.co.za/index.php?option=com_content&view=article&id=60414:Process-key-to-successful-software-development&catid=107 [08 November 2012]

Frank J. 2005. The CanMEDS 2005 Physician Competency Framework. Better standards. Better physicians. Better care. Ottawa: The Royal College of Physicians and Surgeons of Canada. <http://meds.queensu.ca/medicine/obgyn/pdf/CanMEDS2005.booklet.pdf> [25 May 2009].

Galorath, D. 2009. 2009 Standish Chaos Report... Software Going Downhill
<http://www.galorath.com/wp/2009-standish-chaos-report-software-going-downhill.php> [27 April 2010]

Gangani, N. T., McLean, G. N. & Braden, R. A. 2004. Competency-based human resources development strategy, Academy of Human Resource Development Annual Conference, Austin, TX, 4-7 March, in: Proceedings, Vol 2: 1111 – 1118.

Garlan, D., Allen, R. & Ockerbloom, J. 2009. Architectural Mismatch: Why Reuse Is Still So Hard. IEEE Software, 26(4):66-69.

Gegick, M. & Williams, L. 2007. On the design of more secure software-intensive systems by use of attack patterns. Information and Software Technology, 49(4): 381 – 397.

Gibbs, G. R. 2002. Qualitative Data Analysis: Explorations with NVivo. Buckingham: Open University Press, ISBN 9780335200849

Godefroid, P., Halleux, P., Nori, A. V., Rajamani, S. K., Schulte, W., Tillmann, N. & Levin, M. Y. 2008. Automating software testing using program analysis. IEEE Software, 25(5):30-37.

Gottschalk, K., Graham, S, Kreger, H. & Snell, J. 2002. Introduction to Web services architecture. IBM Systems Journal, 41(2): 170 - 177

APPENDICES

- Gravetter, F.J. & Forzano, L. B. 2009. Research methods for the behavioral sciences. 3rd ed. California: Cengage Learning.
- Hailpern, B. & Santhanam, P. 2002. Software debugging, testing and verification. IBM Systems Journal, 41(1):4-12.
- Hofmann, H. F & Lehner, F. 2001. Requirements engineering as a success factor in software projects. IEEE Software, 18(4): 58 – 66.
- Hofstee, E. 2006. Constructing a Good Dissertation: A practical guide to finishing a Master's, MBA or PhD on schedule. Johannesburg, South Africa: EPE.
- Hook, P. 2000. Training for Cyber-War. Computer Fraud & Security, 2000(10): 14 – 15
- Hughes, B. & Cotterell, M. 2006. Software Project Management. 4th ed. Berkshire: McGraw-Hill.
- Hull, E. Jackson, K. & Dick, J. 2005. Requirements Engineering, 2nd ed. Springer. Books24x7: Apress. http://common.books24x7.com/book/id_16286/book.asp [10 June 2009].
- Jiang, L., Eberlein, A., Far, B.H & Mousavi, M. 2008. A Methodology for the selection of requirements engineering techniques, Software and Systems Modelling 7(4): 303 – 328.
- Jick, T.D. 1979. Mixing Qualitative and Quantitative Methods: Triangulation in Action. Administrative Science Quarterly, 24(4): 602-611.
- Jonasson, H. 2008. Determining Project Requirements. Auerbach Publications. Books24x7: Apress. http://common.books24x7.com/book/id_26418/book.asp [18 June 2009]
- Jones, C. 2004a. Quantitative and qualitative research: conflicting paradigms or perfect partners? In Proceedings of Networked Learning 2004. Lancaster, UK, 5-7 April.
- Jones, C. 2004b. Software Project Management Practices: Failure versus Success. CROSSTALK The Journal of Defense Software Engineering, 2004:5-9.
- Juristo, N., Moreno, A.M., Vegas, S. & Solari, M. 2006. In search of what we experimentally know about unit testing. IEEE Software, 23(6):72-80.
- Kendal, S. 2011. Object Oriented Programming using C#. Simon Kendal & Ventus Publishing. ISBN 978-87-7681-814-2
- Klein, M. & Ralya, T. 1990. *Analysis of Input/Output Paradigms for Real-Time Systems, An* (Technical Report CMU/SEI-90-TR-019). Pittsburgh: Software Engineering Institute, Carnegie Mellon University. <http://www.sei.cmu.edu/library/abstracts/reports/90tr019.cfm> [July 18, 2010].
- Knight, W. 2005. Security — built-in or bolted-on to the SOA? Infosecurity Today, 2(2): 38 – 40.
- Kofman, S. 2006. Effects of Virtual Communities on Young Adults <http://www.eden.rutgers.edu/~skofman/myspace.doc> [28 October 2008]
- Kory, D., Chasser, A. & Fingerhut, E. 2006. Full circle: A Report On Technology Transfer In Ohio. http://www.ipu.uc.edu/file_PDF/TTOCFy06.pdf [25 March 2011]

APPENDICES

- Krauss, S. E. 2005. Research paradigms and meaning making: A primer. *The Qualitative Report*, 10(4):758-770.
- Krueger, C. W. 1992. Software reuse. *ACM Computing Surveys (CSUR)*, 24(2), 131-183.
- Kujala, S., Kauppinen, M., & Rekola, S. 2001. Bridging the gap between user needs and user requirements. In *Proceedings of PC-HCI 2001 Conference*. Patras, Greece, 7-9 December.
- Kumar, R. 2005. *Research methodology – a step-by-step guide for beginners*. 2nd ed. California: Sage.
- Le Deist, F. & Winterton, J. 2005. What is competence? *Human Resource Development International*, 8(1): 27-46.
- Leymann, F. & Altenhuber, W. 1994. Managing business processes as an information resource, *IBM Systems Journal*, 33(2):326-48.
- Lowy, J. 2007. *Programming WCF Services*. California: O'Reilly Media Inc.
- Lucia, A. D. & Lepsinger, R. 1999. *The art and science of competency models: Pinpointing critical success factors in an organization*. San Francisco, CA: Jossey-Bass.
- Mack, L. 2010. The Philosophical Underpinnings of Educational Research. *Polyglossia*, 19: 5– 1
- Mackenzie N and Knipe S, 2006, Research dilemmas: Paradigms, Methods and Methodology, *Issues In Educational Research*, Vol 16, <http://www.iier.org.au/iier/6/mackenzie.html> [June 12, 2010].
- Malan, R & Bredemeyer, D. 2002. *Software Architecture: Central Concerns, Key Decisions*. http://www.bredemeyer.com/pdf_files/ArchitectureDefinition.PDF [01 November 2009].
- Malan, R. & Bredemeyer, D. 2009. *Software Architecture and Related Concerns*. <http://www.bredemeyer.com/whatis.htm> [01 November 2009]
- Malik, D. S., 2002. *C++ Programming: From Problem Analysis To Program Design*. Massachusetts: Thomson Learning
- Malik, D.S. 2004. *C++ Programming: From problem analysis to program design* 2nd ed. Massachusetts: Course Technology, Thomson Learning Inc.
- Martin, C. R. & Martin, M. 2006. *Agile Principles, Patterns and Practices in C#*. New Jersey: Prentice Hall
- Mathew, V. 2012. Code Cleanup. <http://oi.vettukal.com/2012/07/week-6-report-code-cleanup-and-delete.html> [07 October 2012]
- McGibbon, S. 2007. Service Oriented Architectures and Web Services – Interoperability by Design. *Baltic IT&T Review*, 37:39-42.
- McGraw, G. 2002. Building secure software: Better than protecting bad Software. *IEEE Software*, 19(6):57-59.

APPENDICES

- Minani, D. & Parker, M. B. 2009. The use of Facebook for social networking within a community on the Cape Flats. Cape Town: Cape Peninsula University of Technology, 91- 105. ISBN 978-0-620-44199-5.
- Mirabile, R.L.1997. Implementation planning: Key to successful competency strategies. Human Resource Professional, 10(4), 19-23.
- Mlitwa, N. 2009. Investigation of Integration and use of ICT for Teaching and Learning in South African Higher Education Institutions; PhD Thesis, University of Cape Town. Cape Town.
- Morrison, M. & Morrison, J. 2003. Database-driven web Sites. 2nd ed. Boston: Thomson.
- Mouton, J. 1996: Understanding social research. Pretoria: Van Schaik.
- Mumford, E. 2006. Researching people problems: some advice to a student. Info Systems, 16(2006):383-389.
- Neuman, W.L. 1997. Social research methods: qualitative and quantitative approaches. 4th ed. USA: Viacom.
- Nieman, G & Bennett, A. 2006. Business management: A value chain approach. 2nd ed. Pretoria: Van Schaik.
- Nosotro, R. 2003. Soren Kierkegaard: Danish Philosopher and Theologian. <http://www.hyperhistory.net/apwh/bios/b2kierkegaard-soren.htm> [15 July 2011]
- Olsen, W. 2004. Triangulation in Social Research: Qualitative and Quantitative Methods Can Really be Mixed. <http://www.ccsr.ac.uk/staff/triangulation.pdf> [10 March, 2010]
- Olson, D. L. 2004. Introduction to Information Systems Project Management. 2nd ed. New York: McGraw-Hill.
- Ort, E. 2005. Service-Oriented Architecture and Web Services: Concepts, Technologies, and Tools. <http://java.sun.com/developer/technicalArticles/WebServices/soa2/> [18 August, 2008]
- Ozkan, B. C. 2004. Using NVivo to Analyze Qualitative Classroom Data on Constructivist Learning Environments. The Qualitative Report, 9(4): 589-603.
- Pereira, J. P. & Parker, M. B. 2009. An alternative approach to empower citizens in communities with tension. Cape Town: Cape Peninsula University of Technology, 105-120. ISBN 978-0-620-44199-5.
- Pras, A. & Martin-Flatin, J.P. 2007. What Can Web Services Bring To Integrated Management? Handbook of Network and System Administration: Amsterdam: Elsevier.
- Reeves, J.W., 1992. What is Software Design? C++ Journal, 2(2)
- Rivera-Ibarra, J.G., Rodriquez-Jacobo, R., Fernandez-zepeda, J.A. & Serrano-Vargas, M.A. 2010. Competency Framework for Software Engineers. Proceedings of the 23rd IEEE Conference on Software Engineering Education and Training, 9-12 March 2010, Pittsburgh, PA; pp 33-40.

APPENDICES

- Rob, P. & Coronel, C. 2004. Database systems: design, implementation & management. 6th ed. Boston: Thomson.
- Roodt, J & Paterson, A. 2009. Skills Shortages in South Africa: Case Studies of Key Professions. Cape Town: HSRC Press, ISBN 978-0-7969-2273-1
- Rouibah, K, & Al-Rafee, S. 2009. Requirement Engineering Elicitation Methods: A Kuwait Empirical Study about familiarity, usage and perceived value. *Information Management & Computer Security*, 17(3): 192–217.
- Ruxwana, N. L., Herselman, M. E., & Conradie, D. P. (2010). ICT applications as e-health solutions in rural healthcare in the Eastern Cape Province of South Africa. *Health information management journal*, 39(1), 17-26.
- Sajeev, A. S. M. 1994. Some Reusability Exercises in Persistent C*. *International Conference on Computing and Information Proc. ICCI'94*: 1160 - 1175
- Sarantakos, S. (1998). *Social Research*. 2nd edition. Palgrave Macmillan, New York. USA.
- Satzinger, J.W, Jackson, R. B & Burd, S. D. 2004. *Systems analysis & design: In a changing world*. 3rd ed. Boston: Thomson Course Technology.
- Serpell, A. & Ferrada, X. 2007 A competency-based model for construction supervisors in developing countries. *Personnel Review*, 36(4): 585-602.
- Shreyas, D. 2002. *Software Engineering for Security: Towards Architecting Secure Software*. <http://www.dsc.ufcg.edu.br/~jacques/cursos/map/recursos/SoftwareEngineeringandSecurity.pdf> [12 May 2009].
- Singleton, R. A. Jr. & Straits, B.C. 2005. *Approaches to social research*. 4th ed. Oxford: Oxford University Press.
- Sommerville, I. & Sawyer, P. 1997. *A Good Practice Guide*. New York: John Wiley.
- Sommerville, I. 2007. *Software Engineering*. 8th ed. England: Addison-Wesley.
- Sotomayor, B. 2005. A short introduction to Web Services <http://gdp.globus.org/qt4-tutorial/multiplehtml/ch01s02.html#id2564226> [13 August 2008]
- Srivastava, B & Koehler, J. 2003. *Web Service Composition – Current Solutions and Open Problems*. <http://www.zurich.ibm.com/pdf/ebizz/icaps-ws.pdf> [21 August 2008].
- Standish Group International. 2001. *Extreme chaos*. <http://www.scribd.com/doc/10167963/Chaos-Report-2001> [20 May 2009].
- Stary, C. 2002. Shifting knowledge from analysis to design: requirements for contextual user interface development. *Behaviour & Information Technology*, 21(6):425-440.
- Stepanek, G. 2005. *Software project secrets: Why software projects fail*. Books24x7: Apress. http://common.books24x7.com/book/id_12530/book.asp [June 12, 2009].
- Talby, D., Hazzan, O., Dubinsky, Y. & Keren, A. 2006. Agile software testing in a large-scale project. *IEEE Software*, 23(4):30-37.

APPENDICES

Tillmann, N. & Schulte, W. 2006. Unit tests reloaded: Parameterized unit testing with symbolic execution. *IEEE Software*, 23(4):38-47.

Turley, R.T & Bieman, J.M. 1995. Competencies of exceptional and non-exceptional software engineers. *Journal of Systems and Software*, 1995(28): 19-38.

Van Vliet, H. 2007. *Software engineering: Principles and practice*. Chichester, UK: John Wiley.

Viega, J. & McGraw, G. 2002. *Building secure software: How to avoid security problems the right way*. Addison-Wesley.

Wang, H., Huang, J. Z., Qu, Y & Xie, J. 2004. Web services: problems and future directions. *Web Semantics: Science, Services and Agents on the World Wide Web*, 1(3) 309 -320.

Weitzman, E., Miles, M.B. (1995), *Computer Programs for Qualitative Data Analysis*, Sage, London.

Welman, C., Kruger, F. & Mitchell, B. 2005. *Research methodology*. 3rd ed. Cape Town: Oxford University Press.

Wirfs-Brock, R. J. 2009. Principles in Practice. *IEEE Software*, 26(4):11-12.

World Bank. 2008. Facts and figures from the world development indicators 2008. http://siteresources.worldbank.org/DATASTATISTICS/Resources/reg_wdi.pdf [27 May 2010].

Yeo, K. T. 2002. Critical failure factors in information system projects. *International Journal of Project Management*, 20(3):241-246.

Yin, R.K. 2003. *Case study research: design and methods*. 3rd ed. California: Sage.

Zelger, J. & Oberprantacher, A. 2002. Processing of Verbal Data and Knowledge Representation by GABEK-WinRelan. *Qualitative social Research*, 3(2).

APPENDICES

Appendix A: Questionnaire

A software developer competency framework

Dear Software Developer,

While some of the following questions require one answer, other questions may require multiple answers.

Please tick (V) or check on appropriate answers accordingly.

1.) Report and feedback regarding this research will be provided to all software developers who will participate in this research.

Would you like to receive reports and feedback regarding this research?

Yes (If yes, please provide your contact details in the next question)

No

General information

2.) (OPTIONAL) If you would like feedback and report please fill in your contact details in the text boxes below.

Otherwise, proceed with the next question.

	Contact details
Email address	—
Your company	—
Job title	—
Name	—

General information

3.) What is your gender?

Male

Female

General information

4.) What is your age?

Under 21 years

21 to 34 years

APPENDICES

- 35 to 49 years
 - 50 to 64 years
 - 65 years or older
-

General information

5.) What is your race?

- Black
 - Coloured
 - White
 - Indian
 - Other
-

General information

6.) What is your citizenship?

- South African
 - Other African countries
 - European
 - Asian
 - North American
 - South American
 - Australia and Oceania
-

General information

7.) What is your work experience as a software developer?

- 0 to 3 years
 - 4 to 6 years
 - 7 to 10 years
 - over 10 years
-

General information

8.) How long did it take you to become a software developer who performs his/her duties with minimal assistance from other developers?

- 0 to 6 months
- 7 to 12 months
- 13 to 18 months
- 19 to 24 months

APPENDICES

over 2 years to 3 years

over 3 years

General information

9.) Are you are a certified software developer, certified with software training authorities like Microsoft, Oracle and Sun?

No

Yes

10.) What is your highest education qualification?

Certificate

National Diploma

BTech/Bachelor Degree

MTech/Master's Degree

DTech/PHD

Other

General information

11.) What is your current level (position) as a software developer?

Junior developer

Middle level developer

Senior developer

12.) In which province are you coming from?

Western Cape

Eastern Cape

Free State

Gauteng

KwaZulu Natal

Limpopo

Mpumalanga

North West

Northern Cape

APPENDICES

Software development tools

13.) What among these tools (IDE) do you use most of the time?

Please scroll further down for more options

- Microsoft Visual studio
 - Microsoft Share point
 - Net beans
 - Eclipse
 - Dreamweaver
 - WinDev
 - Xcode.
 - ActiveState Komodo
 - JCreator
 - PhpED
 - Notepad
 - MPLAB
 - Other
-

Software development tools

14.) What among these programming technologies do you use most of the time?

Please scroll further down for more options

- C#
- C++
- C
- Ruby
- Pascal
- Delphi
- Visual Basic
- Java
- PHP
- COBOL
- Assembler
- Python
- FORTRAN
- Perl
- Cold fusion
- Java script

APPENDICES

- VB script
 - Ajax
 - ASP.NET
 - HTML
 - XML
 - UML
 - CSS
 - Other
-

Software development tools

15.) What among these database management systems (DBMS) do you use most of the time?

Please scroll further down for more options

- Microsoft SQL server
 - DB2
 - Oracle
 - Sybase
 - PostgreSQL
 - Informix
 - NonStop SQL
 - Microsoft Access
 - MySQL
 - Teradata
 - Ingres
 - Other
-

Software development tools

16.) What among these code version control tools do you use most of the time?

Please scroll further down for more options

- Subversion
- Perforce
- Visual Studio Team System
- Visual SourceSafe
- StarTeam
- Vault
- Concurrent Versions System (CVS)

APPENDICES

- JediVCS
 - Other
 - None (if you do not use any code version control tool)
-

Software development tools

17.) What among these application servers do you use most of the time?

Please scroll further down for more options

- Microsoft server products
- WebSphere
- Apache
- Glassfish
- JBoss
- Jetty
- JRun
- Oracle OC4J
- WebLogic
- SAP Netweaver
- Other
- None (if you do not use any application server)

Software development tools

18.) Your software products are installed or running in which environment?

- Windows
 - Linux
 - Unix
 - Other
-

Software development tasks

19.) Among these software development tasks your duties fall in which groups?

- Analyse user requirements/system requirements
 - Design software solution
 - Code or implement software solution
 - Test software solution
 - Deploy or install software solution
 - Perform software maintenance
-

APPENDICES

Software development skills

20.) While performing your tasks as a software developer, how well do you agree that the following skills are important for you to perform your tasks?

Please scroll further down for more options

	Strongly agree	Agree	Don't know	Disagree	Strongly disagree
Communication skills	()	()	()	()	()
Ability to do self-study and research	()	()	()	()	()
Team work	()	()	()	()	()
Lack of ego	()	()	()	()	()
Reuse of code	()	()	()	()	()
Experience with previous work	()	()	()	()	()
Analytical and thinking skills	()	()	()	()	()
Use of prototype	()	()	()	()	()
Focus on customer needs	()	()	()	()	()
Innovative while dealing with problems	()	()	()	()	()

APPENDICES

Software development skills

21.) What other skills do you possess that helps you perform and accomplish your software development tasks?

Thank You!

Thank you for taking our survey.

Please forward the questionnaire link to other software developers

Department of Information Technology

Cape Peninsula University of Technology

[Website: www.cput.ac.za](http://www.cput.ac.za)

APPENDICES

Appendix B: Introductory letter to the industry

ATTN: Managers, Project managers, Team leaders, Software developers

Date: 07 September 2010

RE: Research on competencies of software developers

Dear Sir/Madam,

A research project is currently underway in the Department of Information Technology at Cape Peninsula University of Technology. The research is on competencies of software developers. The purpose of this research is to establish essential competencies for software developers. Once such competencies have been identified and analysed, they could be used by both industry and educational institutions. This would help to prepare software developers and enable them to develop good quality software systems to meet business demands and challenges.

Your company has been selected to participate in this research because you are directly or indirectly involved with software projects. This research will be conducted by using questionnaires. A website link to access the questionnaire will be given to you soon. On behalf of the Cape Peninsula University of Technology, we would like to request for your assistance to ensure that the link to questionnaire is distributed among your software developers and that the questionnaires are filled. **The questionnaire will take a maximum of 10 minutes to complete online.** The questionnaire has questions about software development tools, tasks and skills. All questions have a list of options and your software developers will be required to select options from a given list of choices.

Rest reassured that as a participant in this research, you have the following rights:

- I. Your participation is entirely voluntary*
- II. The information provided by your software developers will be kept strictly confidential.*
- III. While information provided by your software developers may be included in the research report, under no circumstances will names or any identifying information be included in the report.*
- IV. The research report will be made available to you upon your request.*

Please feel free to contact us should you require more information or any clarification regarding this research.

Lastly, in order to make this research successful, please forward this email to other business companies or organizations in the Western Cape Province.

Thank you for your willingness to participate in this research.

David Minani
Research Student, Dept of IT
Faculty of Informatics and Design
Cape Peninsula University of Technology
Phone: 072 554 9008
Email: dminani@gmail.com

Temuso Makhurane
Research Supervisor, Dept of IT
Faculty of Informatics and Design
Cape Peninsula University of Technology
Phone: 021 469 1136
Email: makhuranet@cput.ac.za

APPENDICES

Appendix C: Letter to CPUT internship coordinators

TO: Internship/Co-operative coordinator

Attn: Mr. Bernardo

Attn: Mr. Bronwyn

Date: 07 September 2010

RE: Research on competencies of software developers

Dear Internship/Cooperative Coordinator,

A research project is currently underway in the Department of Information Technology at Cape Peninsula University of Technology. The research is on competencies of software developers. The purpose of this research is to establish essential competencies for software developers. Once such competencies have been identified and analysed, they could be used by both industry and educational institutions. This would help to prepare software developers and enable them to develop good quality software systems to meet business demands and challenges.

This research will be conducted by using questionnaires. We are in a process to finalize the questionnaire. Since we have good relationship with the industry, we would like all companies that accept our students for internship to be involved in this research. The research is on software developers, we would like all companies that are directly or indirectly involved with software projects to take part in this research project.

We would like you to do the following:

- 1) An introductory letter will be sent to you soon, please forward this letter to any business company or organization that is directly or indirectly involved with software projects. This includes all companies involved with the internship programmes with CPUT.
- 2) A website link to access the questionnaire will be given to you soon. Please forward this link to all companies for distribution among software developers. **The questionnaire will take a maximum of 10 minutes to complete online.**
- 3) Please forward the above mentioned to any IT company that you feel should participate in this research.

Feel free to contact us should you require more information or any clarification regarding this research.

Your assistance is highly appreciated for the success of this research project.

Regards,

David Minani
Research Student, Dept of IT
Faculty of Informatics and Design
Cape Peninsula University of Technology
Phone: 072 554 9008
Email: dminani@gmail.com

Temuso Makhurane
Research Supervisor, Dept of IT
Faculty of Informatics and Design
Cape Peninsula University of Technology
Phone: 021 469 1136
Email: makhuranet@cput.ac.za

APPENDICES

Appendix D: Email to software developers

TO: Business organizations and IT companies

Attn: Software developers

Date: 07 September 2010

RE: Research on competencies of software developers

Dear Software developer,

A research project is currently underway in the Department of Information Technology at Cape Peninsula University of Technology. The research is on competencies of software developers. The purpose of this research is to establish essential competencies for software developers. Once such competencies have been identified and analysed, they could be used by both industry and educational institutions. This would help to prepare software developers and enable them to develop good quality software systems to meet business demands and challenges. This research is conducted by using questionnaires. **The questionnaire will take a maximum of 10 minutes to complete online.**

Please do the following:

1. Click on this link: <http://www.surveygizmo.com/s3/357491/A-software-developer-competency-framework> to respond to the questionnaire.
2. Then forward this link to other software developers so that they may also participate in this research.

Feel free to contact us should you require more information or any clarification regarding this research.

Your assistance is highly appreciated for the success of this research project.

Thank you,

David Minani
Research Student, Dept of IT
Faculty of Informatics and Design
Cape Peninsula University of Technology
Phone: 072 554 9008
Email: dminani@gmail.com

Temuso Makhurane
Research Supervisor, Dept of IT
Faculty of Informatics and Design
Cape Peninsula University of Technology
Phone: 021 469 1136
Email: makhuranet@cput.ac.za

APPENDICES

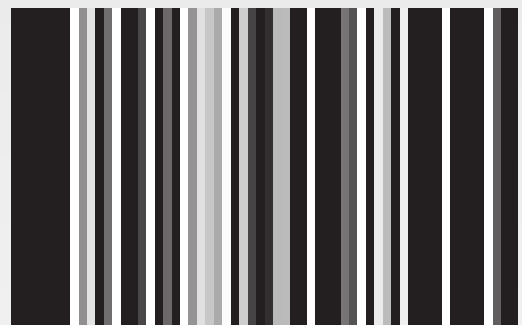
Appendix E: List of companies requested to participate in this research

NO:	COMPANY
1	24.com
2	4cit Software Solutions
3	ABSA Bank
4	Ackermans
5	Actaris
6	Adept
7	Airborne Consulting
8	Alchemy Software
9	Altech ISIS
10	Ananzi
11	APL Cartons
12	Application Frameworks
13	ASHTECH
14	ATIO
15	BP
16	BSG
17	Business Connexion
18	Buzz-IT
19	Capitec Bank
20	City of Cape Town
21	Clicks
22	Clicks Group
23	CPUT
24	dataX
25	DVT
26	e-globe Technologies
27	Elemental
28	Entelligence
29	Eskom
30	Fontera
31	Foschini
32	Fuzzy Logic
33	GE
34	Health Focus
35	Hetzner
36	HubIT
37	ICG
38	iDRIVE
39	idu Software
40	IMMIX Solutions
41	Imperatech Solutions
42	Imperative Informatics
43	Infosense
44	Intec Telecom Systems
45	Iplex Technologies
46	iTouch
47	Keerapa Consulting Services
48	Khanyisa Real Systems
49	Lemon Zest Consulting

APPENDICES

NO:	COMPANY
50	Lianolink
51	Liquid Thought
52	Maxxor
53	MCS Computer Services
54	Media24
55	Metropolitan
56	Metropolitan Health Group
57	MWEB
58	Nebula
59	Nedbank
60	Neotel
61	Net1 Group
62	NHA
63	Old Mutual
64	Open box software
65	Paracon
66	Paradigm Solutions
67	Pennypinchers
68	Peralex
69	Pick n pay
70	Pioneer Foods
71	PSG Treasury Outsourcing
72	Risk Methods & Solutions
73	ROI Media
74	S1 Corporation
75	Samo Consulting
76	Sanlam
77	Shell
78	Shoprite
79	Silicon Overdrive
80	Silverminute
81	Software Futures
82	SPI Group
83	Synergy Nine
84	Telkom
85	TF Design
86	Traderoot
87	Truworths
88	UCS Software
89	Vodacom
90	Webafrica
91	WineMS
92	Woolworths
93	WorkPool

Software Developer Competency Framework



David M. Minani
November 2013