

Towards an ontology-driven software development approach: An unended quest

by

Nehemiah Mavetera

Submitted in fulfilment of the requirements for the degree

Doctor of Philosophy in Information Technology

in the

Faculty of Engineering, Built Environment and Information Technology

University of Pretoria

Pretoria, South Africa

10 November 2011

Candidate: Nehemiah Mavetera

Promoter: Prof. J.H. Kroeze

Department: School of Information Technology

Degree: Doctor of Philosophy in Information Technology

Declaration

I, Nehemiah **Mavetera**, declare that:

Towards an ontology-driven software development approach: An unended quest which I hereby submit for the degree of Doctor of Philosophy in Information Technology at the University of Pretoria, is my own work and has not previously been submitted by me for a degree at this or any other tertiary institution.



Signature of candidate

This 10th day of November, 2011.

Preface

“There is no royal road, but there is a road”

Fredrick, P. Brooks, 1987

It is common knowledge that software development projects more often than not, have missed their schedules, are out of budget or result in flawed products. A plethora of software development methods have been proposed and used. These can be classified abstractly as structured methodologies, object-oriented methodologies and, lastly, as a more recent invention, agile methodologies.

To support these methodologies, several methods, techniques and tools have been developed and used. All these were roads that were developed and used in an attempt to get to the end, but they are not royal roads. As is common in a methodological scientific way of solving a problem, the first step leads to the second, and the future step should be preceded by the current step. All these methodologies have brought the software development process to its current state and it is at this stage that we need to evaluate the successes achieved so far and, in certain cases, the failures too. In the end, we need to find a software development approach and methodology that suits the requirements of the future.

‘Towards an ontology-driven software development approach: An unended quest’ is a research study motivated by the lack of return on investment in current information systems. The root cause of this lack of return on investment is attributed to the way in which software products are developed. Software practitioners have adopted and used industrial engineering principles to fashion software products. Unlike the process of fashioning machines, the fashioning of software products requires an appreciation of the human element that is always present and resident in organizational information systems.

Failure to incorporate the humanistic element in the development process will lead to the resultant products being mechanistic, rule-based and no better than machines. A new road that fosters a transition from the development of mechanistic software products to the development of romantic software products is proposed in this thesis. Briefly, romantic systems are implemented using romantic software products. These are systems that imitate the way human beings behave in organizational information systems. They understand the organizational culture, context, capture tacit and intuitive knowledge among other things. Starting from the classical definition of ontology as a study of existence, the growth and incorporation of such ontology to new uses that focus on *“What is there that we can*

individuate and characterize?”, a new software development approach is proposed. This research study presents a new framework that can be used to develop software products that exhibit human behavioural characteristics and that are both adaptable and evolvable.

Abstract

Towards an ontology-driven software development approach: An unended quest

Nehemiah Mavetera

Over the years the field of software development has undergone a series of mutations, particularly in the types of approaches and methodologies that are applied during the development process. One thing that has not been fully achieved by software development method engineers is to move the development process completely from the mechanistic functionalist paradigm to a neo-humanist romantic paradigm. Although many researchers claim to have introduced new development paradigms in software development, these are merely new methodologies that are grounded in old paradigms. There are three fundamental development approaches that lie in the hard systems approach: the traditional structured, object-oriented and the recently invented agile approaches that have been widely adopted by software practitioners. Few of these practitioners have embraced the soft systems approach and their development methods have not migrated from the syntactic processes of the hard-systems approach.

Another problem that software developers continue to face is a lack of a method or tool that can augment current syntactic programming language technologies and software development methods by the addition of semantic-based tools to facilitate the construction of romantic, adaptive and evolvable software products. In fact, most of the problems encountered in software development can be attributed to deficiencies in the methodologies, that is: the approaches, methods, techniques and tools used during the development of the software product. This research study introduces the concept of ontologies in software development and motivates for an ontology-driven approach to software development that reduces the mechanistic nature of software products but increases their adaptability and usability.

Although current industrial and academic research has focused at the semantic properties of ontologies in software development, researchers have not considered how the methodological process can be designed and used to develop romantic software products. This research study used one variant of GTM and followed an interpretive approach in the investigation of several issues that are known and documented but not addressed by the current software development approaches. The field of software development has been investigated and a framework of requirements that enables the development of romantic

systems is presented. The ontology discipline, focusing on the semantic, pragmatic and contextual characteristics of ontologies, was also consulted. Starting from a set of differentiated ontological frameworks and from the syntactic, semantic and pragmatic nature of ontologies, the research then presents a framework of ontologies that can be co-opted into a software development approach to address the deficiencies in current software development approaches highlighted in the framework of software development requirements.

As part of the research findings, a new definition of ontology, as well as a framework of components that make up the ontology and a theoretical translation model that is needed to develop romantic software products, are presented. The theoretical translation model comprises three parts: an ontology-driven software development framework, an ontological approach to software development and an ontology-based methodology for software development. Of note is the use of domain, method, process, intentional, and status ontologies at different stages of software development to cater for the semantic, pragmatic and contextual gaps that are not currently addressed by existing development approaches. However, in this study, a balance is reached between addressing the needs of current and future developers of software products, that is, one that reacts to an urgent market need, as well as addressing a software development approach need that is heavily grounded in the softer, neo-humanist paradigm.

KEYWORDS

Software development, Framework, Approach, Methodology, Romanticism, Ontology.

To

My Late Parents, Nehemiah (Snr.), Gamuchirai and Wife, Tsungai Mavetera

and

My Lovely Wife, Chipo Gertrude and my children

Samantha, Patience, Kudzai Rejoice, Tinotenda and Kudzanai Nehemiah (Jnr.)

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to my wife, Chipo Gertrude, for her support during this trying time. I certainly could not have got this far without her unwavering support, both at home and work. I will not forget my children, Samantha, Patience, Kudzai, Tinotenda and Kudzanai. I will never forget the number of times you asked about the chapter I was working on.

I would also like to thank my promoter, Professor Jan Kroeze, for his insightful reflections, mentoring and encouragement. It was a great privilege to work with you. I learnt of many professional issues that had been always been elusive from my lower and junior studies. I cannot forget my Head of Department: Professor Carina De Villiers, I say thank you very much.

I cannot forget my workmates. Thank you for allowing me to go on study leave and covering for my absence. It is time for you to get the benefits of my short absence from work. My thanks are also extended to the North West University and to the National Research Foundation (NRF) of South Africa; both these institutions co-funded this research study. Without their financial support, I certainly would not have been able to undertake this study.

To all my interviewees, my sincere thanks. Without your cooperation, I certainly would not have discovered anything. I wish there were more people like you who were willing to provide information for the sake of academic development and research.

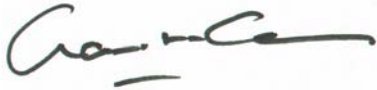
And, lastly,

Thank You, Almighty God, for With You Nothing is Impossible.

Matthew 19, verse 26

This thesis, *Towards an ontology-driven software development approach: An unended quest*, by Mr. Nehemiah Mavetera, was edited for language, grammar and style by Ciaran Michael Mac Carron MA, Hons. BSc, who has been a member of the South African Translators' Institute for over 25 years and has had over 25 years' experience in the editing and translation of theses, manuals etc.

Signed



C M Mac Carron

20 August 2010

TABLE OF CONTENTS

CHAPTER ONE	1
INTRODUCTION AND RESEARCH MOTIVATION	1
1.0 Introduction	1
1.1 Research Motivation.....	2
1.1.1 Communication Requirements in Software Development	5
1.2 The Research Interest	6
1.2.1 Gap in the Field of Study	7
1.2.1.1 The Software Development Process	9
1.2.1.2 Software Development and Ontologies	10
1.3 Research Methodology	11
1.3.1 GTM	11
1.4 Generic Research Propositions	12
1.5 Preliminary Research Questions.....	13
1.5.1 Main Research Question	14
1.5.2 Secondary Research Questions	15
1.5.2.1 The ‘What is?’ type of questions.	16
1.5.2.2 Why ontology?	19
1.5.2.3 The “How should?” And “How does?” types of questions	21
1.5.3 Aims and Objectives of the Study	21
1.5.3.1 Aims of the Study	21
1.5.3.2 Objectives of the Study	22
1.6 Research Scope and Delimitation.....	23
1.6.1 What this Research Study is Not	24
1.7 Contribution of Research to the Body of Knowledge	25
1.8 Structure of the Thesis.....	26

1.9	How to Read this Thesis	28
1.10	Summary.....	29
CHAPTER TWO		31
UNDERSTANDINGS IN INFORMATION SYSTEMS RESEARCH.....		31
2.0	Introduction	31
2.1	What is theory?	32
2.1.1	General Types of Theory	33
2.2	Strategies for Conducting Research	35
2.3	Types of Research in Information Systems	35
2.3.1	Theoretical Research	36
2.3.2	Empirical Research	36
2.4	Research Tasks in Information Systems.....	36
2.4.1	Constructive Research Task	37
2.4.2	Nomothetic Research Task	37
2.4.2.1	Deductive Theory	37
2.4.3	Ideographic Research Task	38
2.5	Sociological Paradigms Applied to Information Systems	38
2.5.1	The Functionalist Paradigm	40
2.5.2	The Interpretive Paradigm	41
2.5.3	The Radical Structuralist Paradigm	41
2.5.4	The Neo-Humanist Paradigm	42
2.6	Philosophical Groundings in Information Systems Research	42
2.6.1	Ontological Grounding to Information Systems Research	42
2.6.1.1	Objectivism	43
2.6.1.2	Constructionism (Constructivism)	44
2.6.2	Epistemological Grounding to Information Systems Research	44
2.6.2.1	Positivism versus Anti-Positivism	45

2.6.3	The Humanistic Grounding in Information Systems Research	45
2.7	Information Systems Research Paradigms	46
2.7.1	Quantitative Research Paradigm	46
2.7.2	Qualitative Research Paradigm	47
2.8	Approaches to Information Systems Research.....	48
2.8.1	The Positivist Stance	48
2.8.2	The Interpretive Stance	49
2.8.3	The Critical Realism Stance	50
2.9	The Nature of the Research Problem.....	51
2.9.1	Ontological Diagnosis	52
2.9.2	Epistemological Diagnosis	52
2.9.3	Humanist Diagnosis	52
2.9.4	Methodological Stance	53
2.10	Summary.....	53
CHAPTER THREE		55
RESEARCH METHODOLOGY AND DESIGN		55
3.1	Introduction	55
3.2	Research and Methodological Approach.....	56
3.2.1	Research Approach	56
3.2.2	Methodological Approach	56
3.2.2.1	Qualitative Research Methods	57
3.3	Research Method Used in this Study.....	59
3.3.1	GTM	60
3.3.2	The Research Design	60
3.3.2.1	Requirements for Sound Application of GTM	60
3.3.2.2	Problem Statement and Use of Research Questions	62
3.3.2.3	Use of Literature	63
3.3.3	Data Acquisition Methods	65

3.3.3.1	Qualitative Data Acquisition	65
3.3.3.2	Data Acquisition Methods for this Study	66
3.3.3.3	Primary Data Acquisition and Sampling	67
3.3.3.4	Secondary Data Acquisition Methods	68
3.3.4	The Research Goals	69
3.3.5	Data Analysis in GTM	71
3.3.5.1	GTM Coding	71
3.4	The Double Mapping Principle	75
3.5	From Substantive to Formal Theory	77
3.6	The Reflexive Grounded Theory Process.....	77
3.7	Judging the Fidelity of Generated Grounded Theory	79
3.7.1	Confirmability	80
3.7.2	Dependability/Auditability	81
3.7.3	Internal Consistency	81
3.7.4	Transferability	81
3.8	Evaluation of Qualitative Interpretive Research Study	82
3.8.1	The Fundamental Principle of the Hermeneutic Cycle	82
3.8.2	The Principle of Contextualization	83
3.8.3	The Principle of Interaction between the Researcher and the Subjects	83
3.8.4	The Principle of Abstraction and Generalisation	84
3.8.5	The Principle of Dialogical Reasoning	84
3.8.6	The Principle of Multiple Interpretations	85
3.8.7	The Principle of Suspicion	85
3.9	Limitations of the Research Methodology	85
3.10	Evaluation Criteria for the Generated Theory	86
3.11	Summary.....	88

CHAPTER FOUR.....	90
SOFTWARE DEVELOPMENT PRACTICES, INFORMATION SYSTEMS AND ORGANIZATIONS.....	90
4.0 Introduction	90
4.1 The Nature of Organizational Systems.....	92
4.1.1 The Complex Nature of Organizational Systems	92
4.1.2 Culture in Organizations	93
4.1.3 The Role of Concepts in Information Systems Development (ISD)	94
4.1.4 Context in Information Systems	96
4.2 The Practice of Software Engineering.....	97
4.2.1 The Software Development Problem	100
4.3 Software Product Development Practices	102
4.3.1 Communication in Software Development	103
4. 3.2 Software Engineering Reference Model	107
4.4 The Need for a Conceptual Schema	109
4.5 The Mechanistic Nature of Information Systems.....	110
4.5.1 Mechanistic Systems, Systematicity and System-Formation	113
4.5.1.1 Romanticism	114
4.5.2 Transition to Romantic Systems	115
4.6 Activity Theory, Actor-Network Theory and Theory of Organized Action	117
4.6.1 Activity Theory	117
4.6.2 The Social Network Aspect of Activity Systems	119
4.6.3 Human Activity Systems (HAS)	120
4.6.4 Theory of Organized Activity (TOA).	121
4.7 The Approach versus Methodology Debate	122
4.8 Software Development Approaches	126
4.8.1 Structured Development Approach	126

4.8.2	Object-Oriented Approach (OOA)	127
4.8.3	Agile Approaches	127
4.8.4	Goals of a Software Development Approach	129
4.9	Model Requirements for a Romantic Software Product.....	130
4.10	Software Development Issues	131
4.11	Summary.....	133
CHAPTER FIVE		134
HISTORY AND NATURE OF ONTOLOGY		134
5.0	Introduction	134
5.1	What is Ontology? A Brief Description	135
5.2	Background to Ontology Applications	137
5.3	Other Views of Ontology	138
5.4	Ontology Perspectives	140
5.4.1	Architectural Characteristics of Ontology	140
5.5	The Concept of Conceptualization	142
5.6	Varieties of Ontology	143
5.6.1	Domain Ontologies	144
5.6.2	Method Ontologies	145
5.6.3	Status Ontologies	145
5.6.4	Intentional Ontologies	145
5.6.5	Social Ontologies	146
5.6.6	Process Ontologies	146
5.7	Possible Ontology Uses in Software Development.....	147
5.7.1	Ontologies and Communication	147
5.7.2	Ontologies in Modelling	148

5.7.3	Ontologies in Distributed Non-homogeneous Database Systems	148
5.7.4	Ontologies in Systems Integration	149
5.7.5	Ontologies and Software Products Reusability	150
5.7.6	The role of ontologies in a Romantic Information System	150
5.8	Ontology-Driven Software Development Architecture.....	153
5.8.1	The Knowledge Base Repository	154
5.8.2	The Designer Engine	154
5.8.3	The Reasoner	155
5.9	The Field of Semiotics and Ontology.....	156
5.9.1	Syntactics	156
5.9.2	Semantics	157
5.9.3	Pragmatics	158
5.9.4	Social Aspect (Context)	159
5.10	Towards a Working Definition of Ontology	159
5.11	Summary.....	162
CHAPTER SIX.....		164
DATA ANALYSIS AND THE ONTOLOGICAL FRAMEWORK.....		164
6.0	Introduction	164
6.1	Profile of Interview Respondents	165
6.1.1	Thematic Areas Considered During Data Gathering	166
6.1.2	Data Sampling and Preparation	167
6.2	Components of the Data Analysis Tool Atlas.Ti.....	169
6.3	GTM Coding Issues.....	169
6.4	A Practitioner’s Classification of Software Development Aspects.....	175
6.5	Propositions and Research Questions Revisited.....	177
6.5.1	Refinement of Propositions	179

6.5.1.1	The Software Development World View	180
6.5.1.2	The Software Development Paradigm	180
6.5.1.3	The Software Development Approach	182
6.5.1.4	The Software Development Method	185
6.5.1.5	Adaptive and Evolving Software Development	187
6.5.1.6	The Software Development Environment	188
6.5.2	Refined Research Questions	190
6.6	GTM Theoretical Sampling and Saturation.....	192
6.6.1	Propositions PA and PB Story Lines	192
6.6.2	Proposition PB Story Line	193
6.6.3	Proposition PC Story Line	193
6.6.4	Proposition PD Story Line	195
6.6.5	Proposition PE Story Line	195
6.7	The Double-Mapping Principle Revisited.....	197
6.8	A Requirements Framework for a Software Development Approach-The What Part.....	198
6.9	The Ontology-Driven Software Development Framework–The How Part.....	202
6.10	An Ontology-Driven Software Development Approach-The How Should Part.	207
6.11	Discussions	211
6.12	Summary.....	212
CHAPTER SEVEN		213
RESEARCH EVALUATION, CONCLUDING STATEMENTS AND FUTURE WORK.		213
7.0	Introduction	213
7.1	Research Quality Considerations.....	213
7.2	Contribution to the Field of Software Development	215

7.3	Evaluation of the Study’s Contribution to the Body of Knowledge.....	218
7.3.1	Evaluation of the Generated Theory	219
7.3.2	Evaluation	226
7.4	Recommendations, Future Work and Limitations.....	227
7.5	Conclusion	229

BIBLIOGRAPHY230

APPENDICES253

Appendix A: Data Gathering Interview Questions	253
Appendix B: Publications Generated from the Research Process	261
Appendix C: Incidents for Proposition PA	264
Appendix D: Incidents for Proposition PB	265
Appendix E: Incidents for Proposition PC	267
Appendix F: Incidents for Proposition PD	270
Appendix G: Incidents for Proposition PE	273
Appendix H: Full list of interview audio recordings (on CD)	275
Appendix I: Full list of interview transcripts (on CD)	275
Appendix J: Full list of incidents and codes for all the interviews (on CD)	275
Appendix K: Network diagram (on CD)	275
Appendix L: Full List of codes and code families for all the interviews (on CD)	275
Appendix M: Full list of incidents and memos for all the interviews (on CD)	275
Appendix N: Definition of Concepts Used in the Thesis	275

TABLE OF FIGURES

Figure 1.1: Preliminary Research Questions (<i>Adapted from Roode, 1993</i>).....	15
Figure 1.2: Structure of Thesis.....	27
Figure 2.1: Sociological Paradigms (<i>Adapted from Burrell and Morgan, 1979</i>)	40
Figure 3.1: Three-Tier Incident and Concept Mappings	76
Figure 3.2: A Reflective Grounded Theory Process (<i>Adapted from Gasson, 2003</i>)	78
Figure 4.1: Communication Gap in Software Development	105
Figure 4.2: Software Reference Model (<i>Adapted from Gunter et al., 2000</i>).	108
Figure 4.3: Components of an Integrated Information System (<i>Adapted from Sowa, 2000</i>)	109

Figure 4.4: The Concept of an Activity System (<i>Adapted from Roque et al., 2003</i>).....	118
Figure 5.1: Different Realms of the World (<i>Adapted from Goldkuhl, 2002</i>)	138
Figure 5.2: Structural Representation of Ontology.....	141
Figure 5.3: OntoSoft Case Tool Architecture (<i>Adapted from Mavetera, 2007</i>)	153
Figure 5.4: A Linguistic Model (<i>Adapted from Molenaar, 1998</i>).....	160
Figure 6.1: Aspects of Consideration in Software Development	177
Figure 6.2: Expanded Double Mapping Principle	198

LIST OF TABLES

Table 3.1: Research Goals and Data Gathering Methods	70
Table 3.2: Criteria for Judging Qualitative-Interpretive Research	80
Table 4.1: Characterizing Approach, Methodology and Method	125
Table 4.2: Philosophical Groundings of Software Development Approaches (<i>Adapted from Brown et al, 2004:4139</i>)	128
Table 6.1: Interview Themes	167
Table 6.2A:Categories from Open Coding Process.....	172
Table 6.2B: Thematic Areas Identified from Open Coding Process	173
Table 6.3: Software Development Relationships from Practitioners' Perspective.....	176
Table 6.4: Requirements Framework for a Software Development Approach	199
Table 6.5: Ontology-Driven Software Development Framework	205
Table 6.6: An Ontology-Driven Approach to Software Development.....	208
Table 6.7: An Ontology-Driven Software Development Methodology	209

CHAPTER ONE

INTRODUCTION AND RESEARCH MOTIVATION

1.0 Introduction

“...So many books are produced to meet an identified market opportunity rather than written because an argument presses to be expounded.”

Peter Checkland, 1999.

Much research has been done to address a market-related need quickly, to the detriment of the soundness and quality of the knowledge generated from such an endeavour. Although many market research projects are carried out within specific paradigms it is rare to find research projects that are intended to improve on or establish new paradigms in specific fields. Instead, the results from market-driven research usually create loopholes in their application areas, rather than eliminating these. This is because the grounding paradigms have been overtaken by new market requirements or societal changes. Market-driven results are often portrayed as comprehensive models, frameworks, architectures or as tangible artefacts but the most overarching problem is their lack of comprehensiveness. Comprehensive information system (IS) products are fundamentally based on the generally accepted design principles of systems engineering (Gerber, 2006).

In the present research, a balance is reached between addressing the needs of current and future information systems, that is, reacting to a market need as well as addressing a development approach need that is heavily grounded in the softer, neo-humanist paradigm. Checkland (1999) noted that, although many research projects claim to be introducing a new development paradigm in software development, they are merely new methodologies that are grounded in old paradigms. Checkland (1999) fell victim to this when, in an attempt to move the system development approach to a new paradigm, grounded in the soft systems approach, he was unable to outline a holistic, methodological approach that minimized the influence of the hard systems approaches after the analysis stage. In his case, after he employed a softer approach during analysis, the design and implementation stages were left to rely on the mechanistic dictates of the hard system paradigm. In his soft systems approach, the design and implementation stages lack the softer elements captured during the earlier analysis stage.

As will be noted in Chapters 4, 5 and 6, there are a plethora of software development approaches that are grounded in two development paradigms: the hard systems and soft systems development paradigms. The traditional development approaches, that is, the structured approach, the object-oriented approach and the agile approach, are heavily inclined to the hard systems paradigm. The only difference between these approaches is the methodologies they use.

Hence this research fills the knowledge gap in the fields of information systems and software development through the development of an ontology-driven approach to software development that is heavily grounded in the soft systems paradigm. The present research, therefore, explores and contributes to an approach to software development that has evolved in the research community but which is only sporadically adopted by software development practitioners. The major reason for its being sporadically tested and applied in practice is the lack of guiding principles in the use of this method. The proposed approach is based on an ontology-driven software development framework that lists a set of ontologies that should be developed and used in each phase of the development life cycle. This is an attempt to link the natural world, the social world together with the artificial world of human constructions during software development. As Gregor (2006) noted, information systems can only be understood if theory that links these three elements is found. Currently, many theoretical constructs that are used in the development of IS artefacts do not address this holistically. IS researchers are therefore encouraged to pull together the three disciplines of the natural sciences, social sciences and design sciences in order to address the requirements of IS development problems.

1.1 Research Motivation

“There is a reason why computers have not yet become fervent natural language speakers. (It’s not a matter of processing power and never will be): we simply are not programming them correctly.”

El Baze, 2005.

The field of software development is characterized by mechanistic development practices that denote the software development field as rational and deterministic. Consequently, when the resultant software products are implemented in information systems, these systems exhibit a mechanistic character that eventually limits their usability. Mechanistic systems are based on the concept of explicit programming (Agentis, n.d.). Explicit

programming produces software products that do not capture semantic or context-rich data, characteristics that are needed in most modern systems. However, these systems are very efficient in structuring data to enable and facilitate its interpretation.

Mechanistic practices also overlook the notion that, as representatives of human principals, information systems should forge some type of humanistic and non-deterministic behaviour. This behaviour can only be captured in the software product.

Some examples of mechanistic practices in systems development manifest themselves in products such as the conceptual schema concept (Sowa, 2000), requirements, design and implementation specifications. This conceptual schema is widely used in database management systems and knowledge bases.

Some mechanistic practices also manifest themselves as software development approaches, such as the structured approach (Pressman, 2005), object-oriented approach (Pressman, 2005, Dennis *et al.*, 2002), the software product lines approach (Carnegie Mellon Software Institute, n.d.) and the software kernels approach (Dittrich & Sestoft, 2005).

Software kernels (Dittrich & Sestoft, 2005) and software product lines (Carnegie Mellon Software Engineering Institute, n.d.) have been used in industry, research and academia in an attempt to speed up the process of reproducing and developing software products. These rely heavily on the speed of program generation which is an example of a mechanistic explicit systems implementation strategy. Although they have gained widespread acceptance, they still leave a semantic gap in software products that needs to be filled. Many of these software development approaches have worked in the past but are starting to fail in the current organizational environment and could be worse in the future. This can be supported by Trim and Lee (2004:478-479) who noted that “the models, concepts and theories” that have been used in software development in the past cannot be used to solve peoples’ development problems in the future. This, therefore, necessitates research in new approaches to software development.

Several attempts have been made to reduce the mechanistic tendencies of software products. Harris *et al.* (2009) proposed controlled flexible approaches that allow flexibility in the development process but which are not too prescriptive, such as the waterfall approach, widely known as the plan-driven approaches. They argued that every software development approach requires some controls and that future research has to

look at the controls that should be included in the development methodology and at how they should be introduced. The research should indicate whether they are outcome or emergent outcome controls (Harris *et al.*, 2009). Wand and Weber (1990:63) strongly advocated for a system development process that would capture “structure (statics) and behaviour (dynamics) of the real world”. As they argued, to achieve this, researchers must use ontologies to provide both “sufficient human-oriented and machine-oriented” descriptions of real world systems (Wand & Weber, 1990:63). Whitten *et al.* (2004) encouraged system developers to design their systems for growth and change. Pressman (2005), in discussing some software myths, raises issues such as the ability of software to evolve, quality measurement, throughput levels, reusability of software products, management of scope creep during development and documentation of software products, as some of the most misunderstood and difficult aspects that need to be handled carefully during a software development process. This misunderstanding stems from the fact that a socially constructed organizational system is forced onto a machine-like environment through the process of deconstruction, forcing it to lose some of its life states, hence losing its human elements.

‘Towards an ontology-driven software development approach: An unended quest’ is a quest for finding and defining an “effective problem representation” (Hevner *et al.*, 2004:83). The approach discussed herein, as a construct can prove to be very crucial for the effective design of software solutions by software developers.

The Evolutionary Software Paradigm

Unlike software artefacts that are static, organizations are dynamic and always in a continuous state of change (Kawalek & Leonard, 1996). This posits software products at a difficult position to represent the true state of an organization. Legacy systems are, because of these reasons, difficult to adapt and maintain. Kawalek and Leonard (1996) and Meso and Jain (2006) argued that software products should be innovative, adaptive and replicable. Naturally change is endemic in organisations; therefore new methods of developing software products that exhibit these characteristics should be formulated. For this reason, Kawalek and Leonard (1996:189) advocated for development methods and practices that produce “instantly adaptable software that is able to support radically changing demands on a series of fast developing platforms and integrating with a series of end user developments”.

The Software versus Organizational Paradox

Kawalek and Leonard (1996) discussed a paradox where software products hinder organizational change and progress. They reason that a holistic, real world system is forcibly embedded in a piece of software as a representation of an organization. If viewed in conjunction with Kant's Philosophy of Deconstruction, the piece of software is always a partially understood description of the status of an organization at any one particular point in time. Unlike software products, organizations are always in a state of flux (Dahlbom & Mathiassen, 1993; Kawalek & Leonard, 1996) which constantly change and duplicate in their existing environments. There is therefore a need for developers to match the static nature of software products, the dynamic nature of the organization together with the dynamic nature of the software model. Lehman (1991) noted that software products have to be evolutionary if they are required - and are able - to solve problems in the real world domain. This is however complicated by the fact that evolving systems rely on the organizational context, which itself is dynamic and a software product development process that is uncertain.

Current software products can be classified as the static type (S-type). These static type products stress that the correctness of a software design is the only criterion for the success of a software product, coupled with its fidelity, "in a strict mathematical sense" to the specification (Kawalek & Leonard, 1996:189). Because of this, software products cannot move from one steady state to another steady state if they are to reflect and implement what occurs in organizations. In other words, software products cannot be optimized against static goals since organizations are not static. All these arguments point to the fact that software development requires a different approach than the current hard approaches. The new approach should embrace the socio-technical nature of software products and recognize the duality of organizational context and that of developing software products. Besides the software and organizational paradox discussed herein, the software development field also faces several other problems, some of which can be heavily attributed to the communication methods used during development.

1.1.1 Communication Requirements in Software Development

Most of the problems encountered in software development can be attributed to deficiencies in the communication methods (*Section 4.3*), methodological processes, i.e.

the approaches (Section 4.8), techniques and tools used during the development of the software product. Stakeholders in software development, such as customers, analysts, designers and programmers, are faced with a serious problem of communication during software development. Modelling notations and languages such as data flow diagramming (DFD), entity relationship diagrams (ERD) and unified modelling language (UML) tends to be too technical and varied to be understood by system users. At times even analysts have problems of choosing a communication tool that can be understood by programmers. Contrary to most common beliefs, unified modelling language is not as unifying a communication language among all stakeholders as most developers and its inventors would have wanted (Dobing & Parsons, 2007). A real unifying language or communication tool is needed to lessen the burden of communication during software development.

Another problem is that, during system specification, many software development tools, modelling tools or case tools do not fully comply with the software development specifications. Examples of this are the UML specifications. Of the tools available, many have minimal application or use in big projects (Dobing & Parsons, 2007). Instead, Dobing and Parsons (2007:125) call for a development tool “that provides 15% of what is needed” in terms of integrating the different development tasks and providing a uniform, common language interface for all stakeholders. This implies that, of the available tools, none can provide a functionality that is more than 15% of the basic requirement of a software development environment (SDE). The nature and form of the development tool that should accompany any system development approach has therefore “a considerable influence on how the underlying methodology will be used” (Dobing & Parsons, 2007:125). Hence the choice of such a tool is very important in the development of pieces of software. The next section discusses the researcher’s interest in this field.

1.2 The Research Interest

“The problem is that data is trapped in hierarchical silos, restricted by structure, location, systems and semantics. The situation has become a data graveyard.”

Sheryl Torr-Brown (In-PharmaTechnologist.com, 2005.)

A global information society requires data, information and knowledge to be freely accessible and shareable. Despite being archived in disparate information silos, an exchange medium, or obligatory passage point should be introduced that allows

unrestricted access to this data by different users in different organizations (Introna, 1997; Mavetera, 2004b). The medium could be a component of information systems. In a follow up to these requirements, research has focused on the development of the Semantic Web (W3C). This is a software development project that endeavours to provide documents and services that have meaning to the user. This meaning is captured as formal descriptions (meta-information) (Lemmens, 2006) lying on a platform of machine processable knowledge structure referred to as ontologies. Ontologies, its touted, as a medium in information systems, help to improve the capturing of the semantic content and of the pragmatic contexts of information systems.

However, the problem that continues to face software developers despite the introduction of ontologies is the lack of a method or tool that can augment current syntactic programming language technologies and software development methods through the addition of semantic-based tools to facilitate the development and construction of software products. Large data, information and knowledge silos exist in organizations but their storage structures, location and the systems that access them continue to be a limiting factor for their accessibility and usability (In-PharmaTechnologist.com, 2005). Organizational systems then become graveyards, as their resources cannot be accessed and used. Because of this a methodology for building information systems has to be developed that encourages sharing and adds meaning and context to the data and information.

1.2.1 Gap in the Field of Study

In industrial and academic research, as new problems arise, researchers have to face the challenge of finding solutions to these problems. As Basden (2001) explains, the problems that limit the usability of information systems and the lack of return on investment from these investments should be tracked back to the approaches, methods and processes that are used at the time the “*artefact*” is developed. This artefact is the software product.

It is the software product that is used to implement all the three basic (software based) technological components of an information system, that is, the database, user interface and applications (Sowa, 2000). At the same time, researchers also realized the importance of changing the syntactic nature of software products and, subsequently, of information systems. They thus focused their efforts on finding techniques and tools that could be

used to change the syntactic nature of these products. However, they forgot to re-examine the way in which software products are developed.

This research has as its foundation, the requirement for reasoning to use concepts. Concepts are captured and manipulated by means of machines. At the same time they have a situatedness that is defined and bounded by the context of the situation. This approximates the way in which humans communicate and reason in organizations. It is for this reason that, Lemmens (2006) motivates for a paradigm shift in the development of information systems. He argues that current systems are too data-centric and that developers should move to functional centric methods of developing systems. At a functional level, since processes capture the behavioural aspects of systems, their focus will neglect the use of instances of data as system behavioural representations.

Data-centric systems are mechanistic. Their mechanistic nature can be attributed to the greater emphasis placed on the technical aspects of the systems development process than on the softer management side of this process (Checkland, 1999). In support of this, Hohmann (2007) and Beynon *et al.* (2008) advocated for the development of intuitive systems that are easily understood by humans at the same time increasing the productivity gains from their use. Beynon *et al.* (2008) argued further that marrying together intuition and software development has been made difficult because researchers and developers use frameworks that are inherited from the computer science discipline. These frameworks concentrate on “stable contexts of experience that can be engineered to exhibit law-like characteristics” (Beynon *et al.*, 2008:4) and they do not allow some degree of freedom in cognition (Tarnas, 1991). The developers of mechanistic systems have concentrated on abstracting and representing organizational systems patterns and interactions and uses that can be automated.

In order to address the mechanistic nature of systems, industrial and academic researchers have focused their attention on the semantic properties of ontologies. This again is proving to be another futile exercise, because they are not considering how the ontology properties could be introduced methodologically in the development of software products. This is widely supported by Isabella (1990), who noted that many research studies focus on the design and development of concrete and observable aspects of ontologies in information systems, but that very few have paid attention to the identification and understanding, the interpretations and cognitions associated with the use of ontologies in information systems. Also, despite all the efforts to use ontologies in information

systems, there is still no clearly defined purpose of IS ontologies. This is blamed on the lack of a clear distinction between philosophical ontologies and IS ontologies. Zúñiga (2001:188) believes that IS discipline is “either not equipped to advance general ontologies or (is) not employing the right methodology or theoretical approach” to the use of ontologies in this field. In short, despite these discussions, there is very little evidence of research on the methodological implications of introducing ontologies in software development. The present study takes this as its departure point and will try to address this.

1.2.1.1 The Software Development Process

Gonzalez-Perez and Henderson-Sellers (2006) characterize the software development process as a complex activity in which people, technologies and their organizations are participants. The effects of these participants on the software development process and the use of the software development products are discussed in *Section 4.6.3: Human Activity Systems*. In addition, there are various types of stakeholders in software development, whose diversity contributes greatly to the complexity and difficulty of developing software products. To add to the stakeholder diversity, there is no common vocabulary or world view that can be understood and used by all participants in the development process. The requirement of such a vocabulary is discussed in *Sections 4.8.4, 4.9 and 5.7* of this thesis.

On the software side, the development of software products and information systems has been bedevilled by the lack of a development approach that captures the human aspect of organizational systems. This problem is deeply embedded in the sociological development paradigms that are adopted in the analysis of the problem area that needs to be addressed by the subsequent software products. Depending on the paradigm adopted, whether functional, humanistic, radical structuralism or interpretive (Burrell & Morgan, 1979) (*see Section 2.5*), the resultant analyses of the problem area will differ. At present, many software products are analyzed using the functionalist paradigm, a notion that has seen a plethora of mechanistic software products being developed and implemented. Strongly attached to this paradigm is the use of the reductionist dogma that also supports systematicity and system formation principles (*see Sections 4.5 and 4.5.1*).

This development approach, while addressing the data-processing requirements of most organizations, fails to address the dynamic nature of these organizational systems. Being

themselves dynamic, organizations require dynamic systems to accomplish their tasks and achieve their information-processing goals. Dynamic systems are best modelled using predicative models, since these allow for “formal analysis of system behaviour” (Lemmens, 2006:144). In the context of this study, a task is an action performed in an attempt to accomplish “a goal in a particular context” (Lemmens, 2006: 144) and a goal is an intended achievement by an actor, such as a system or a person.

With the advent of the Internet and global networking infrastructure, ubiquitous and pervasive computing system environments have grown very large. This also extends the social domain that needs to be addressed by software systems. A socially grounded ubiquitous and pervasive environment such as this requires software products that can be adapted to the change in the processing and social requirements that are imposed by the demands of such a massive processing web.

At the same time the social aspects of computing environments require a shift in the paradigm that is consulted during the development of a software product. It is therefore essential that researchers and software developers come up with a development approach that is enshrined in the philosophy of interpretive or humanist sociological paradigms. This is supported by Kirlidog and Aytol (2010) who saw a pressing need for new methodological approaches to software development that could improve the quality of the software products.

1.2.1.2 Software Development and Ontologies

With the advent of ontologies and their use in software development, many researchers have quickly moved to incorporate these artefacts in the production and use of software products.

On the ontology research side, work has been done on ontology development, representation languages: OWL, DAML and Protégé. Tools for ontology building and mapping, semantic mark-up languages and mark-up of resources such as web services have also been developed. On the interoperability side, semantic interoperability frameworks (Mavetera, 2004b; Mavetera, 2007 and Lemmens, 2006) have also been suggested, some of which have been implemented. Several uses for ontologies in conceptual modelling have also been discussed (Wand & Weber, 1993; 2002; Sugumaran & Storey, 2002; Shanks et al., 2003).

1.3 Research Methodology

Every research project needs to be guided and informed by a specific paradigm, ontology and epistemology of the research area (*See Sections 2.5 and 2.6*). These three aspects will enable the researcher to understand the nature of the research problem and ultimately to plan one's methodological stance. The methodological process, like the three aspects of paradigm, ontology and epistemology, exhibits a dichotomous relationship (Fitzgerald & Howcroft, 1998) between qualitative and quantitative methodologies. At this point, the researcher needs to choose the research method that is best suited to the nature of the research problem at hand.

By definition, a research method is a strategy of inquiry which moves from the underlying philosophical assumptions of the research problem to research design and data collection. The choice of research method influences the way in which the researcher collects and analyses the data. Specific research methods also imply different skills, assumptions and research practices (Myers, 1997). The present study is of a qualitative nature. As such, Grounded Theory Method (GTM), the method used in this research, requires the researcher to start without any preconceived theories about the problem to be investigated. However, as will be explained in Chapter 3, preliminary research questions were used in this study. A brief description of this method is given in Section 1.3.1 and the method is fully discussed in Section 3.4.

1.3.1 GTM

Many people confuse grounded theory, i.e. the inductive theory generated using GTM with the methodology (GTM) itself. The GTM is an inductive type of research method that seeks to develop theory that is grounded in data, that is, grounded theory (GT) (Glaser & Strauss, 1967; Glaser, 1992). According to Olivier (2004), GTM starts by observing the field of interest and theory is allowed to emerge from (is grounded in) what is observed in the data. It is important that any bias in data collection be limited to the barest minimum, by the researcher desisting from reading literature in the substantive area of research, starting with a problem statement or hypothesis, hence the research questions as advocated by Glaser (1992).

Research data are then collected systematically and analyzed as and when they are gathered. According to Cornford and Smithson (1996), these data can be used to develop (induce) the final hypotheses, propositions, themes and classifications as the study

progresses. Also, as an inductive, theory discovery methodology, GTM “allows the researcher to develop a theoretical account of the general features of a topic while simultaneously grounding the account in empirical observations or data” (Martin & Turner, 1986:141).

From the analysis of the first data samples, Glaser (1992) encourages the researcher to look for an emerging problem and theory. This becomes the tentative or preliminary theory, which will be adjusted continuously as new data are gathered and analyzed (Cornford & Smithson, 1996; Olivier, 2004). The basic tenets of GTM call for a continuous interplay between data collection and analysis.

GTM is extremely useful in developing context-based, process-oriented descriptions and explanations of the phenomenon (Orlikowski, 1993). A process of purposive sampling is the key to the success of this method, which can only be stopped if saturation level is reached. Saturation level is defined as the point at which the addition of new data from interviews or observations does not change the quality of the knowledge already gathered (Glaser & Strauss, 1967; Glaser, 1992; Olivier, 2004). In GTM, theory is something that shows itself after the collection and analysis of some or all of the data associated with or relevant to the research problem (Bryman, 2004). Lastly, the reader is reminded that GTM frees the researcher from the bondage of longstanding assumptions (Suddaby, 2006), such as starting with a problem statement and the like.

As a qualitative type of research method, research data are best gathered using data-gathering strategies such as interviews, questionnaires, analysis of Internet documents and literature surveys. In this study, literature surveys and interview strategies were employed. The data analysis was done using a text-analysis software product called Atlas.Ti. For a more comprehensive discussion of the data gathering and analysis processes, the reader is referred to Sections 6.1 and 6.4.

1.4 Generic Research Propositions

In support of Glaser and Strauss (1967) and Glaser (1992) that one must not start with hypotheses, this research started with preliminary propositions and, hence, preliminary research questions were also developed. These propositions can be supported from the background readings of Sections 1.1 and 1.2 above.

Proposition A:

The field of software development needs a framework that can be used in the development of romantic software products.

Proposition B:

The software development process can be improved by using an ontology-driven approach to software development.

These two propositions are grounded in the fact that the development of organizational information systems has long been declared to be “*a crisis*”. Despite several attempts at improving the methodological process of developing these systems, little has yet been achieved. Researchers have directed their attention to the way in which software products, the aggregations of which make up systems, are being developed. Attention is being paid to the paradigm requirements of addressing socio-technical problems such as organizational systems. Ontologies are portrayed as facets that can address this paradigmatic problem. The framework requirements should, therefore, consist of ways of including ontology components in software development. These two propositions are further refined in Chapter 6, after the GTM process of open coding.

GTM does not encourage researchers to presumptuously propose aims and objectives of the research. However, based on Strauss and Corbin (1990)’s dicta, preliminary research questions are proposed herein. These questions emanate from the research interest highlighted in Section 1.2 and the above preliminary research propositions.

1.5 Preliminary Research Questions

Most research consists of what investigators refer to as the main and minor research questions. The answer to the main research question, if comprehensively dealt with in the study, will meet all the requirements of the investigation. The main research question can be broken down into several smaller questions whose sum total is equivalent to the single main question. In this study, these smaller questions are referred to as minor research questions. The success of any investigation depends on the ability of the researcher to formulate and answer the research questions that are derived from the problem statement, in this case, the research interest. This section is primarily concerned with the development of these preliminary research questions. The final research questions are discussed in Chapter 6 after initial data analysis.

In the present research the GTM was used as a methodology of study. In an attempt to align the research process with the soundness of the research method chosen and in compliance with its methodological dictum, the final research questions were formulated but only after the collection and analysis of three initial data samples. The preliminary research questions are also necessitated by the need to guide, bind and direct the research (Strauss & Corbin, 1990), as well as to prevent it from spiralling out of context and scope. As the proponents of grounded theory, Glaser & Strauss (1967) would have liked, and as was strongly emphasized by Glaser (1992) in Basics of Grounded Theory Analysis, the final research questions, were allowed to emerge from the data.

1.5.1 Main Research Question

From the research propositions, the main purpose of the present research was to find an approach, a framework of components that could be used to improve the development of software products. The research questions are meant to guide the scope of the investigation using GTM. Unlike many research studies that present a single main research question, this study has two main research questions, covering the software development field and ontology field as separate substantive areas respectively. The main questions to be addressed, therefore, are:

‘What are the components of a software development approach that can be used to develop romantic software products?’

AND

‘What is the role of ontologies in the development of romantic software products?’

In ubiquitous and pervasive environments such as the web, developers currently use intelligent software agents. These agents have gained much popularity in organizational information systems, in which they traverse different information silos in search of information that satisfies their human principals’ request. Examples of these are the electronic market systems (Mavetera and Kadyamatimba, 2003). In these systems, application programs and databases are all linked together to form a large network that can be traversed and searched by software agents. Ontologies have found uses in the development of knowledge bases to complement these databases or application packages. This increases the communicative, semantic and contextual capabilities of these systems.

This research investigated the role that ontologies can play in improving the software development process and data awareness in systems, in adding meaning to information systems and in improving the pragmatic aspects of impact and knowledge in information systems. This objective was achieved by focusing on how the software product as an artefact is currently being developed.

1.5.2 Secondary Research Questions

In order to formulate the secondary preliminary research questions, the Process Based Research Framework (PBRF) developed by Roode (1993) was used in this study. The PBRF, as a tool for developing research questions, gains its validity in its ability to capture the ontological, phenomenological, epistemological and normative characteristics of a research problem. It is generally agreed that each problem can consist of four generic research questions. These are shown in Figure 1.1 as the ‘what?’, ‘why?’, ‘how should?’ and ‘how does?’ type of the problem statement.

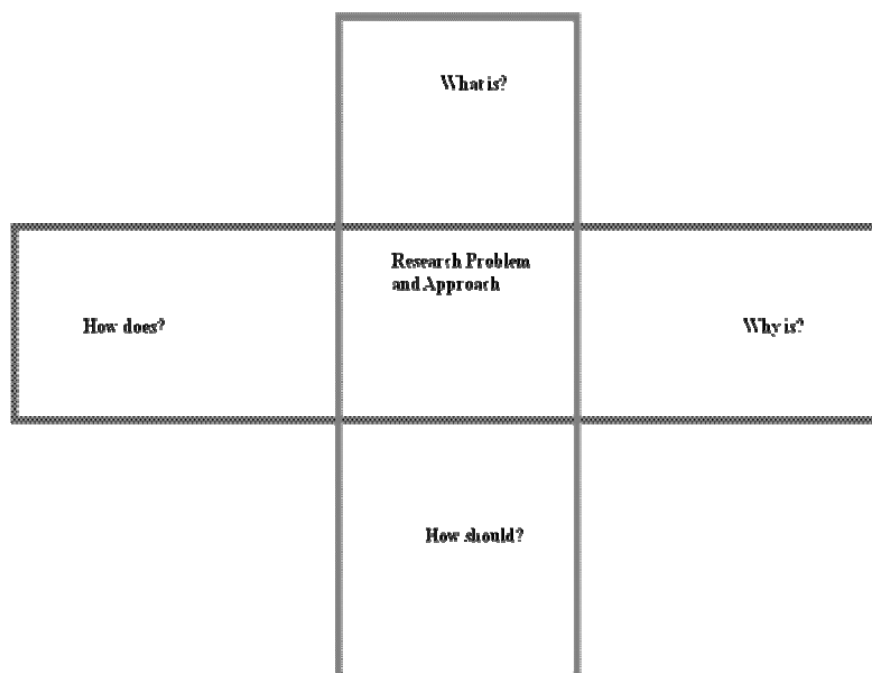


Figure 1.1: Preliminary Research Questions (Adapted from Roode, 1993)

1.5.2.1 The ‘What is?’ type of questions.

These types of questions explore the fundamental nature or essence of the research problem. When these questions are answered, the structure of the problem is made explicit and the ontological and epistemological aspects of the concepts used in the research are discussed. A critical and radical assessment of the problem domain and the underlying paradigms is the focus of such type of questions. The existential proof of the problem is also explored. In this study, the following questions were identified:

What is a mechanistic software product?

This research was driven by the need to improve the usability and adaptability of current information systems. It is, therefore, necessary to discuss the current methods of information system development that lead to mechanistic system development. Mechanistic information systems can be described as systems that rely on - and are based on - the ability of computer systems to represent and arrange the signs that are used in a language. These systems can relate one sign to another but they do not give the meaning of these signs to the user. The systems allow efficient and effective computation and structuring of data but still remain very syntactic. Good examples of these systems are the popular electronic data interchange systems (EDI) used in electronic business systems. As mentioned earlier, these do not provide meaning and context to the user of what has been processed. The user requires another cycle of interpretation in order to understand the output from these systems. In Tarnas’ (1991:266) words, the material particles captured and represented in these systems possess “neither purpose nor intelligence”.

In this thesis, the notions of explicit programming that have been in widespread use in the fields of software engineering are discussed. The researcher then discusses the more accommodating methods that use ontology and lead to romantic systems development.

What is a romantic software product?

In this study, romantic information systems (RIS) are not only limited to the dictates of syntactic machine representations. They are also based on the romantic world view that considers the “world as a unitary organism” in sharp contrast to the rational atomistic view of the mechanistic world (Tarnas, 1991:366-7). The idea for romanticism in information systems is supported by Hohmann (2007:18), who calls for a “pluralisation of our culture and the humanization of technology”. In his vision, he sees a future that demands technologies that stimulate creativity and inspire thoughts, thereby reconciling

the “contradictions between technology and art” (p.18) that characterize the modern era. The ontological approach discussed in this thesis captures the requirements of implicit programming that lead to reusable, context-aware software products. Along the way, the research justifies a methodological approach to software development needed in the development of romantic software products.

The notion of romantic software products as touted in this study requires some elucidation. These products should possess human and behavioural characteristics as those found in organizations where humans play a central role in task management. The research will, therefore, use the following working definition of romantic systems:

‘Romantic information systems are socially-constructed systems that capture and maintain the softer elements of organizational systems such as culture, social context, and semantics and to an extent pragmatics.’

To borrow Beynon *et al.* (2008:5)’s diction, romantic information systems (RIS) should have a balance of the formal methods evident in classical computer science and the informal indeterminate pragmatic methods that “focuses on the complexity of the sociological processes that surround software practice and experience and invokes a more human centred approach”. It must be noted that the current researcher is possibly the first to coin these systems ‘romantic information systems’. However, a romantic information system can be viewed as one that has “gloried in the unbounded multiplicity of realities” (Tarnas, 1991:368) that are realised in organizations as a result of the subjectivity and the divergence in perspectives of the people found in them. Romantic systems should accept the notion that “reality is constructed by the mind, not simply perceived by it, and many such constructions are possible [...]” (Tarnas, 1991:396). These systems should instead capture and reflect all the possibilities in organizational life states (intensions) than to concretize a single life state only as a fact.

What would a Romantic Information System look like?

This section uses an agent-mediated electronic-market system (e-market) as discussed in Mavetera and Kadyamatimba (2003) as an example to illustrate the concept of a romantic information system. In an e-market system, there are seven stages (phases) as derived from the consumer buying behaviour and the business-to-business transactions models whose transactions need to be automated using software agents. These are need identification, partnership formation, supplier brokering, negotiation, contract formation,

purchase and delivery and lastly, product service evaluation, what many organizations may refer to as ‘after-sales service’ (Mavetera and Kadyamatimba, 2003:160).

Central to the success of e-market systems is the database schema that is used to structure the information and data that is exchanged in these systems. However, if the data schema of one participant in the market system changes, most, if not all the participants’ schema need to be changed to conform with these new changes. This becomes very critical in e-market systems where big players trade with small business. This process affects the software development stacks that make up the architecture of the e-market distributed web-based system. This process besides being laborious is also very expensive especially to the small players in the e-market system. Finally, this may limit the participation of small businesses in the market place.

As an example, during the negotiation process, software agents require a shared understanding of the negotiation process. Hence, a negotiation protocol is needed that is understood by all the agents involved. In a romantic e-market system, there must exist several negotiation protocols that can cater for most if not all the participating agents’ requirements such as rules and parameters of negotiation. This can be realised as a negotiation template that captures all the possible intended negotiation environments (negotiation intensions) that can be anticipated in that specific industry. A single negotiation protocol may limit the participation of agents that are not designed for that specific protocol. Also, the negotiation parameters can either be constrained or open but like a human mind, should not remain static during the negotiation process.

Consistent with the definition of romantic systems given above, as representatives of human principals, software agents must therefore possess organizational culture, should have a shared understanding of the product space (a repository of all the products in the market) and should understand the requirements of other agents (shared meaning). Most importantly, they should understand the organizational social context and finally, they should be able to plan, do, check and act decisively as would their human principals. This is a measure of intuition and pragmatics. An agent, for example, can decide to quit a negotiation process at any time if other agents bring unfavourable terms. This can be done after balancing the terms of the negotiation process with its organizational culture and context. This example will be revisited again in chapter 5 to highlight the role of ontologies in building romantic information systems.

What is ontology?

Ontology has attracted many definitions but the one that seems to be widely adopted portrays ontology as a formal specification of a shared conceptualization (Guarino, 1998; Gruber, 1993; Studer, 1998). Guarino *et al.*, (1994:2) also regard it as “knowledge about a priori structure of reality”; implying that, when one chooses a particular intended model for a logical theory, one has to make “implicit assumptions about other models that are compatible with the chosen one”. In information systems, ontology can be regarded as a “formal language designed to represent a particular domain of knowledge” (Zúñiga, 2001:187). In this field, it plays a functional role, since it is almost always designed for specific purposes.

In the study, the philosophical and epistemological aspects of ontology were dealt with in answering this question (*Section 5.1*). Despite its widespread use in philosophy as an essence of existence, there is no unanimous agreement in the information systems field on what ontology is. There are differing schools of thought according to whether ontology is portrayed as a data, information or knowledge artefact (Mavetera, 2004b). Furthermore, its structural components, semiotic requirements and representation in information systems also constitute a subject of debate in this research. All these issues will be addressed in Chapter 5 of this thesis.

1.5.2.2 Why ontology?

There have been several calls for new methods of developing software products with recently, Cretu (2010) proposing a business process oriented software methodology and Bensta (2010) advocating for a software development method that merges ontologies with currently “existing methods, techniques and tools” that are used during the analysis phase. Bensta (2010) further argues that the merger can improve significantly the process of software development at all stages of analysis through to maintenance, by facilitating the faithful translation of user requirements into object models that are used to develop the system specification. Mavetera (2007) proposed the Ontosoft framework that also, like Bensta (2010) framework, positions ontologies at the centre of an automated software development case tool. Hofferer (2007) discussed a semantic interoperability approach that combines meta-models and ontologies. In this approach, ontologies complement meta-models by adding semantic expressiveness to business models and this can be important when integrating business models for different organizations. In addition,

ontologies have long been accepted as artefacts that can improve the software process (Falbo *et al.*, 2002) and Wand and Weber (1990:69) say that an “ontological approach to understanding and formalizing information systems concepts provides [...] the rudiments of a theory of the deep structure of an IS”. The lack of such a theory has deeply undermined research in information systems. The rationale for using ontologies in software development is further supported by Sugumaran and Storey (2002; 2006). They noted that an ontology-driven software development environment is quite effective especially when developing entity-relationship diagrams (Sugumaran & Storey, 2006) but they also lamented the lack of organized, systematic methods for developing and using ontologies in information systems (Sugumaran & Storey, 2002).

This research study justified why information systems need to incorporate the ontology artefact during their development (*Chapters 5 and 6*). The architectural and structural aspects of the ontology artefact are discussed and used to justify an ontology-driven approach for software development (*Chapter 5*). Of the two parallel streams, that is, semiotics and formal logic that can be used to describe ontology, the semiotic stance (Sowa, 2000; Stamper, 1992) has been used to build an argument that posits ontology as an artefact that can solve among others, the current problems of linguistic communication during software and systems development (*Section 5.10*) and the capturing and representation of human and softer characteristics of organizations. Since IS ontologies are confined to that which can be represented, it should be noted that subjective things such as feelings, even though they exist, cannot as yet be represented, especially in automated machines. This is not a limitation to the development of romantic software products touted in this research, but is a shortfall attributed to the technology used for representation.

A working definition for ontology that will be used in this study defines ontology as: *a linguistic model of the world that comprises of syntactics, semantics, pragmatics as well as the social context of that which is represented. Despite some unavoidable informal indeterminacy in the real world view, it should allow a shared, descriptive, both structural and behavioural modelling and representation of a real world view by a set of concepts, their relationships and constraints under the open world assumption.*

1.5.2.3 The “How should?” And “How does?” types of questions

This research arrived at a methodological solution to current problems in software development. In this solution set, which is a framework of ontology components, the ontology is positioned as an artefact that can be used during the development of information systems and, at run-time stage, to decrease the semantic gap currently existing in information systems.

This process was realized by taking the preliminary propositions and building them up through a literature survey (*Chapters 4 and 5*) and empirical data gathered during interviews (*Chapter 6*). This empirical study, using GTM, was done in order to develop the ontological framework accepted in the software development field. The framework depicts components of the ontology that can be looked for by any would-be system developer in a bid to make the resultant system romantic.

As the research problem does not focus on the implementation part of the research findings, it has very few technical requirements. It focuses more on the logic, the 'What?' of the problem. This emphasis on the philosophical justification of abstract concepts and ideas makes the ‘How does?’ type of questions less relevant. These types of questions are meant to explore issues directly observable or those that can be described as they manifest themselves in reality (Roode, 1993). At the minimum they look at the development of a design science artefact.

1.5.3 Aims and Objectives of the Study

“In this way, knowledge becomes reusable, visible and accessible and enables teams to share their knowledge and profit from experience.”

Sheryl Torr-Brown (In-PharmaTechnologist.com, 2005).

1.5.3.1 Aims of the Study

This study aims to find a software development approach that can improve the software development process. This comes as a framework of components that should be considered during the software product-development process.

The approach, in addition to focusing on the unavoidable use of syntax, should also find ways of ensuring the capturing of semantics, pragmatics and social context of the

organizational system. Context ensures that the fuzzily defined social requirements of every system are also included in the resultant system.

It is worthwhile to pre-empt the research outputs by advocating the use of the ontology artefact during software product development. The aim of the study is to position ontology as an artefact that can be used during the development of software applications, at run time or as part of an information system, in order to increase semantic and context-awareness in information systems. This development will result in information systems that are more dynamic and usable, with components that are reusable, shareable, visible and accessible to other systems and users as well, especially in agent-mediated environments such as those described by Mavetera and Kadyamatimba (2003). It is the intention of this research to investigate and bring to the fore the humanistic, interpretive and softer aspects of ontologies as used in software development and information systems.

1.5.3.2 Objectives of the Study

In line with the aims of the study, the objectives of the study can be summarized as the need to develop a general ontological framework that satisfies the aims of this study. The ontological approach to software development challenges traditional software engineering approaches as expounded in this research. More specifically the objectives of the study are:

- To develop or construct a tangible ontological theory within the field of software development that can be used to design and develop software products. The abstract ontology theory could be operationalized into a concrete form of knowledge that could be applied readily in software development (Agerfalk, 2004). Operationalization is the process of converting or moving from abstractions to some concrete artefacts that have practical applications in a social practice such as software product development.
- To position ontology as an artefact that is needed by software developers to experientially and intuitively develop the linguistic models needed for communication in information systems.
- To develop an ontology model-driven approach to software development (El Baze, 2005). This approach would ensure that the resultant model moves from a purely

abstract conceptualization of an ontology model to a truly operational model. This context-aware, purpose-specific operational model can be deployed as a value-added service layer in software development processes.

This research, while it also specifically discusses implementation issues, however, placed greater emphasis on software development methods than on the technological tools needed in software development.

1.6 Research Scope and Delimitation

The research purpose was to use socio-technical behavioural theories to explain the social nature of organizations. These organizations need to be represented as information systems using software products. Although several software development approaches are discussed, the discussion is basically to bring to the fore the problems that are embedded in those approaches that need to be addressed. The research is constructivist, focusing on the development of a new software development approach and of a methodology that can guide the development of software products that capture the human aspects of organizations. This is done through a process of identifying persistent software development problems that have not yet been addressed fully by researchers.

The research developed a working definition of ontology as used in the field of information systems. The definition was arrived at by following philosophical and practical paths of arguments as discussed in Chapter 5. The study also investigated current software development issues and concerns through a literature survey (*Chapter 4*) and by means of interviews with software development practitioners (*Chapter 6*). During the literature survey, literature related to software development and engineering, information systems and information systems development methodologies was studied. This was used both as existing and unrelated literature to set a tone for the investigative process. The research then focused on the relationship between the requirements of romantic software products and the capabilities of the ontology artefact (*Chapter 6*). It is however an attempt to provide an approach or methodology, guiding principles or a “specification of the process followed plus the modeling approach” to be undertaken in order to develop romantic software products (Gonzalez-Perez & Henderson-Sellers, 2006:125).

1.6.1 What this Research Study is Not

According to Hevner *et al.*, (2004:84), the objective of IS research is to acquire and create the knowledge and the necessary know-how that enables the design, development and subsequent implementation of IT solutions. These solutions may address previously “unresolved as well as important business problems”. There are two basic types of research strands that have since guided research in IS, namely behavioural and natural science research. Behavioural scientists are preoccupied with the understanding of the phenomena that affects human behaviour while natural scientists focus on natural phenomena in general (Purao *et al.*, 2008). If the research falls in the behavioural science category, it should focus on the “development and justification of theories explaining or predicting phenomena that occur” (Hevner *et al.*, 2004:84). Between the behavioural and natural science research groups we find design scientists. These use knowledge from both natural science and behavioural science to develop “means and prescriptions” to aid human endeavours (Hevner *et al.*, 2004; Purao *et al.*, 2008). The intention of design science research is to construct and build concrete artefacts in the form of a model. Purao *et al.*, (2008), while accepting that the boundaries and contours for design science research are still ill defined and fuzzy, they see it as being prescriptive and in contrast to behavioural science research, which is more descriptive. Among other things, the primary goal of design science research is to improve the working state of an artefact, with understanding of the phenomenon being of secondary importance. Design science research should follow six steps identified by Peffers *et al.*, (2008) as: problem identification and motivation, definition of the objectives for a solution, design and development, demonstration, evaluation and communication. This research will not discuss these different strands of research further but will leave the interested reader to check with the sources cited herein.

However, the distinction between design science and other types of research will assist in clarifying the focus of this study. Several issues are discussed in this research that are not necessarily the focus of this study. Although several issues will be touched on and discussed, this research does not solely:

- Aim to identify problems that trouble software developers in the past, present or in the future;

- Intend to develop ontologies as artefacts, a product of design science research. It should be borne in mind that this research does not intend coming up with a piece of software, model, or a design artefact; or
- Aim at characterizing software development approaches, methodologies and methods that are in current use. Such a characterization can be found in several literature publications on software development, for example, Iivari *et al.* (1998).

The research does not deal with the technical aspects of programming and description logics. This technical part, which involves the actual implementation of components of the approach develop in this study, will form part of a future research project.

This research, however, aims at producing a set of guidelines, a conceptual framework that should inform developers on the artefacts to use when developing software products. In addition, these guidelines are packaged as an approach (at a very abstract level) and methodology (more prescriptive) that can be likened to a software development process model (Schach, 2005) such as the waterfall, spiral or iterative models. Although the ontology artefact proposed here requires a design science type of methodology (Hevner *et al.*, 2004), to develop and implement, the packaging of the approach discussed in this thesis is best addressed using a constructivist type of research method. Hence GTM is the method of choice. Furthermore, while the research presents an approach and a methodology, it does not prescribe a specific process for the construction of pieces of software.

1.7 Contribution of Research to the Body of Knowledge

The research study focused on ontology discipline, a field of science whose body of knowledge is continuously growing and is at times not well documented in books or journals. Consequently, it was very important for the researcher to document all the research activities that were carried out during this study. This documentation manifests itself as the conference and journal publications that are listed in Appendix B. In this list, Mavetera (2004a & 2004b) motivates for the use of ontologies in systems development, information systems and their philosophical understanding. In both these works Mavetera discusses the new paradigm requirements of software development approaches, that is, the move from mechanistic to romantic development approaches. These issues are covered in this chapter as well as in Chapters 4 and 5 of this study. Mavetera (2007) also

discussed an ontology software development case tool architecture that is also included in Section 5.8 of this thesis.

Mavetera and Kroeze (2009a & 2009b) address the mechanistic nature of software products and issues that need to be considered in software development from the practitioners' perspective. These are covered in more detail in Chapters 4 and 6. Mavetera and Kroeze (2008 & 2009c) discuss the methodological facets of GTM that were used in this study. These are captured in Chapter 3 and later applied in Chapter 6 during data analysis.

Also, Mavetera and Kroeze (2010a; 2010b) and Mavetera (2011) together presented the ontology-driven software development framework, approach and methodology that can be used to introduce ontologies in the development of software products. Lastly, Kroeze *et al.* (2011) addresses the need for enriching information systems and in this paper, this author discusses the need for introducing ontologies in development of software products. These outputs could be used in the quest for developing more usable software products and they represent a modest attempt to show the importance of this research study.

1.8 Structure of the Thesis

Some researchers, especially those at PhD level, start with a pre-planned, preconceived and an almost standard way of presenting their research outputs. The presentation in this thesis differs considerably from many theses that the reader may have previously encountered. The structure of this thesis is dictated by (grounded in) the research method employed.

Apart from this chapter, all the chapters in this thesis were arranged and positioned in accordance with the steps that were followed during the research process. In the words of Roode (1993), the structure follows a “process based investigation framework”, in which the sequential arrangement of activities in the research dictated the position of the chapter in which they are described. The arrangement of the activities is in turn dictated by the steps followed when GTM is used as a research tool. The layout of the rest of the thesis is shown in Figure 1.2 and is described below.

As shown in Figure 1.2, Chapters 4, 5 and 6 were dealt with simultaneously and spanned almost the entire research process. In other words, while working on the data collection

and part of the analysis, the researcher also consulted literature on software development and ontologies. The reasons for this are given in Chapters 3 and 6.

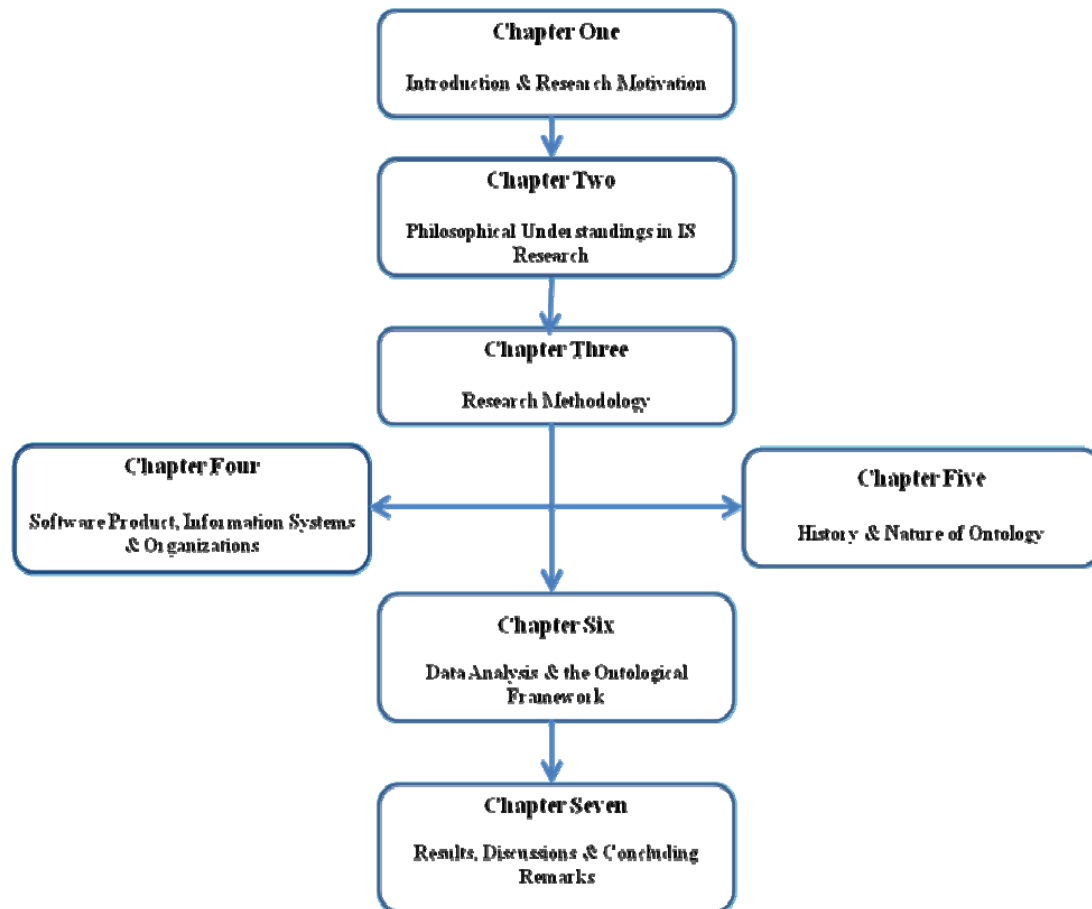


Figure 1.2: Structure of Thesis

Chapter 1, Introduction and Motivation, gives a brief background to the software development problem. The motivation for doing the research, the research interest, a brief description of the methodology used and of the scope of the research is also discussed.

As sound scientific research has to be based on some philosophical grounding, **Chapter 2**, Philosophical Understandings in IS Research, discusses some of the philosophy of information systems research, issues such as the ontological and the epistemological aspects of research and the paradigms of information systems research. The philosophical grounding, that is, the IS Research Paradigms, Approaches and Methods are also included. These are used to guide the researcher in choosing the right research methodology relative to the nature of the research problem.

Chapter 3, Research Methodology and Design, discusses the research method used, data acquisition methods, data analysis techniques and the justification for their selection and

use. It contains a literary discussion of the steps used during the research and the adjustments of the method made, where this deviated from the GTM principles as proposed by Glaser and Strauss (1967). The discipline of information systems development and, hence, of software development is very complex. To support this, **Chapter 4**, Software Development Practices, Information Systems and Organizations, deals with aspects of software development, complex and adaptive information systems, the context of information systems, mechanistic versus romantic systems and the role of actors in modern information systems. It is specifically dedicated to the study of existing literature. The theoretical frameworks that guide the interpretation and analysis of information systems components in organizations are also discussed. These frameworks include - but are not limited to - actor network theory, human activity systems, theory of complexity, deconstruction and philosophy and the software development problem.

Chapter 5, the History and Nature of Ontology, amongst other aspects, discusses the philosophical grounding and the foundation of ontology as an artefact for use in software development. A structural definition of ontology, based on the semiotic levels of Stamper (1992), is also given.

Chapter 6, Data Analysis and the Ontological Framework, discusses data analysis and the ontological approach, which is the output of the study. Its validity in respect of the software development problem is also discussed. **Chapter 7**, Concluding Statements and Remarks, concludes the study and makes some recommendations, proposes quality requirements of the research and suggests the way forward in the field of software development. Also included in this chapter are the research findings, evaluation, limitations and contribution of this research to the field of software product development.

1.9 How to Read this Thesis

Several issues need to be explained concerning the presentation of work in this thesis. These include:

- The use of italics in the thesis body and the proposition sections of the appendices.

In the body of the thesis, some words are italicized to indicate the emphasis the researcher put on those points to express the points under discussion. This emphasis is for the researcher alone and not the original source of the quotations.

In the propositions Section (*Chapter 6*); italics were used to indicate where a set

of words or phrases as collected from the respondents support the proposition listed.

- The referencing style used in this thesis.

The researcher used the University of Pretoria's Harvard referencing style which can be found on the Faculty of Engineering, Built Environment and Information Technology (EBIT) website. For ideas that span the whole text of a chapter, a book or journal paper, the researcher only cited the authors' surnames and the year of publication. In the case of specific quotations, the surname, year and page number(s) were included. In the case of a quotation that followed immediately after another and which was on the same page of an article as the previous quotation, the page number only is included.

- There are many cross-references in this thesis. This represents an attempt to link information together as well as to make it easier for the reader to find the thread in the discussions that fall in different chapters or sections. They also group the thesis discussion as a single body of knowledge. More importantly, the double-mapping principle discussed in Section 3.5 necessitated the mapping of concepts from the software development study discipline to ontology study discipline. This mapping can clearly be shown by using cross-references.
- Usually, most research studies completely discuss their research design on the research methodology chapter only. In this study, the research design straddles two chapters: *Chapter 3: Research Methodology and Design* and *Chapter 6: Data Analysis*. This is not unusual and was necessitated by the requirements and dictates of the research method used, in this case GTM. GTM requires that, as one analyses the data, one concurrently needs to motivate for the choice of the data samples and their coding.

1.10 Summary

This chapter gives an introductory tone to the thesis and its structure. It gives an overview of the aspects covered during the research. More importantly, readers are directed to the issues and problems that have bothered software developers since the software development process was termed a "*crisis*" in 1968 (Randell, n.d.). In Chapter 2 the philosophical underpinnings to information systems (IS) research are discussed. As knowledge is regarded as a social creation, the discussion of these philosophical concepts

about reality will help the reader to gain some understanding of the researcher's world view.

CHAPTER TWO

UNDERSTANDINGS IN INFORMATION SYSTEMS RESEARCH

2.0 Introduction

Chapter 1 gave an introduction to the research study. It gave a road map of the issues that will be covered in subsequent chapters of this thesis. Chapter 2 will give a brief background of some philosophical tenets that are used to guide research in information systems (IS). Research in information systems, as in any scientific discipline, is guided by different philosophical paradigms, research approaches and a myriad of research methodologies. Information systems research is a form of inquiry, an investigation to discover knowledge and facts. Researchers in this field should be convinced (and should be able to convince others) that the knowledge gained from the research process can indeed be relied upon (Olivier, 2004). Most research in information systems should establish actionable knowledge (Agerfalk, 2004) as the research output. Actionable knowledge means the theories, strategies and methods that govern or control the actions performed in social practices by people (Goldkuhl, 1999).

On this note, a research framework is, therefore, needed to guide researchers for them to get actionable knowledge as the research output. The research framework provides answers to questions about the nature of the problem under investigation. This usually forms part of the methodological grounding of the research. The guiding research framework should have as its grounding the philosophical understandings of nature (social paradigms), the guiding research paradigms and the research methodologies. This framework dictates the research approach, the methodology to be used for the research and the subsequent quality of the research process.

Most research tasks usually start with some theoretical considerations. The research is then designed to answer questions posed by those theoretical considerations. Most importantly, the research should be soundly based in theory and undertaken in a systematic way using appropriate research methods. As an output, each research task should either generate or test theories and also provide material for the development of general laws in a discipline.

In addition to this, every research project should build a body of scholarly work in the particular field of study. This scholarly contribution is needed to add new knowledge in a specific study discipline. In this research, the field of study is information systems.

According to Kuhn (1922-1996), scholarly work is judged by sound theoretical

underpinnings that exist in a stable paradigm. The paradigm should be clear to people who share the same thinking in a field of study. These theoretical underpinnings exist also as theoretical frameworks upon which all actions performed in social practices can be soundly based. Continuous investigation or research in a scholarly field may lead to changes in accepted theories; new ideas may gain authority and result in the rejection of old ideas. This may lead to a paradigm shift in the practices of people in the discipline.

This chapter is dedicated to the discussion of philosophical tenets that apply to information systems research. The discussion informs the researcher about the nature of the research problem, the world in which the research is situated and the most appropriate methodological process to follow when solving the problem. As a starting point to this chapter, the general types of theory found in research and the research strategies that are used to generate the theory are briefly discussed.

2.1 What is theory?

“A theory is not some hunch, or half baked idea that you come up with while taking a shower, or being under the influence of something or other.”

Tucker, W., 2009:1

Theory is a term employed in both the science field and day-to-day common usage. It has generated varied definitions from the scientific community and the world at large. In common usage, theory may mean an opinion, conjecture or even speculation. The electronic Dictionary.com regards theory as a coherent group of general propositions used as principles of explanation for a class of phenomena. In a scientific discipline, theory is defined as a logical, self-consistent model or framework that explains the behaviour as a result of the interaction of some related natural or social phenomena. Berg (2007) conceives theory as a set of statements or propositions that are used to describe some aspects of phenomena. Supporting this notion, Sutton and Staw (1995:372-373) see theory as consisting of conceptual arguments, together with logical explanations of these arguments. They further add that, “theory is about connections among phenomena, a story about why acts, events, structure, and thoughts occur.” From the natural sciences perspective, Tucker (2009:9) adds that theory must be a “scientifically tested principle or body of principles that incorporates and explains a significant body of evidence”. In short, theory must express some truth about the relationship between elements of the reality space. Supporting theory as evidence of some truth, Suddaby (2006:633) added that scientific truth “is a product of (empirical) observations and the agreed revelations from the data that are observed by a group of similar minded

observers”. This is supported by Bryman (2004) who also describes theory as an explanation of observed regularities. In many practices, for one to generate or prove theory, data have to be collected to build or test these theories. As Berg (2007:20-21) noted, a theory should, therefore, include “patterns, concepts, process, relationships or events” that are interrelated and are able to explain the being of some occurrences. Furthermore, concepts can either be “symbolic or abstract elements” that are used to signify some special occurrences of phenomena. There is also a need to indicate the “concepts and causal arguments” borrowed from citations and their link or association to the theory under development (Sutton and Staw, 1995:373). These concepts in a theory are generally used as communicative symbols.

It is widely accepted in the scientific domain that any theory generated must be falsifiable, but Sutton and Staw (1995:371) note that the research fraternity lacks consensus on whether “falsifiability is a prerequisite for the very existence of a theory”. This is coupled by the diverse uses of the word in different scenarios, contexts and in different fields of study. Sutton and Staw (1995:371) urge editors of journals to consider research papers that may “test part rather than all of a theory”. This is an acceptance that theory can also be generated by considering only part of a sample size of data or evidence needed to verify or refute a hypothesis from the theory. In addition, Sutton and Staw (1995:375) advise that the research papers can also “use illustrative rather than definitive data”

The fact that theory is able to explain the relationships between natural or social phenomena is not a licence to its acceptance in a study discipline. Quite in agreement with what led to Aristotle’s death when he refuted the geo-centricity of the galaxy, Sutton and Staw (1995:372) note that even well grounded theories, those that fit the empirical data gathered can be rejected if they happen to clash with some “particular conceptual tastes” of the gurus in the field such as editors of journals. The presence and existence of a theory and its subsequent acceptance are not necessarily linked but, it must be noted that, good theory is one that is “representational and verbal” (Sutton and Staw, 1995:376). The next section will discuss some different types of theory found in research.

2.1.1 General Types of Theory

In research practice, there exist generally two types of theory: grand theories and middle-range theories. Grand theories deal with more abstract and general explanations of things (Merton, 1967). Grand theories do not clearly indicate to the researcher a way of collecting

empirical evidence, nor do they provide a clear guide to the researcher on the methods of data collection. This type of theory may not be very useful in a researcher's quest to determine practical relevance of the research.

Merton (1967) also describes middle-range theories as those that are in-between general theories of social systems which are too remote from particular classes of social behaviour, organization and change to account for what is observed and to those in depth orderly descriptions of particulars that are not generalized at all. Mid-range theories are required to guide empirical enquiry and these operate in a limited domain that falls between grand theories and empirical findings. Researchers most often use middle-range theories to understand and explain a limited aspect of social life (Bryman, 2004). However, after noting the ambiguity in the meaning of mid range theories, Gregor (2006:613) warned researchers always to bear in mind that the field or discipline of study influences the "nature of theory" generated by a specific type of research.

In her description of the five types of IS theory, Gregor (2006:612) noted that the "structural nature or the ontological character" of what is termed theory is not well defined, especially in the IS discipline. This ontological nature refers to the "language for talking about the nature and components of theory". She, however, lamented the continual lack of a clear definition of the term theory in many published articles when the term is used more than once in the same article. In her view, whenever an author uses the word theory, the context in which it is used should also be given.

Gregor (2006) just like Hevner *et al.* (2004) and Puroo *et al.* (2008), went on to describe the five types of theory in IS as theories for: analyzing, explaining, predicting, explaining and predicting and, lastly, for design and action. Of importance in this study are the theories of analyzing and explaining, the categories on which this research study is based. In brief, the theory for analyzing classifies characteristics of phenomena by summarizing common attributes that are found in discrete observations. These theories state that the "what is?" of a population sample and the most evident examples of this type of theory are frameworks. Although they have a descriptive element in them, they also analyse and summarize the most important salient attributes of phenomena and their relationships thereof. The most important thing is that the relationships are may not necessarily be causal relationships (Gregor, 2006).

The theory for explaining, also known as theory of understanding (Gregor, 2006), focuses more on the "how?" and "why?" of the existence of some phenomena and their relationships.

In this type of theory, the focus is not on developing testable predictions but simple explanations of “how, when, where and why events” (Gregor, 2006:624) happened are sufficient. These explanations usually give rise to process-type theory. The issue here is to provide new insights as to how nature is, that is, insights that are different from those previously known.

Any research that generates or tests a theory needs to be grounded in a research strategy. The next section will look at the strategies that are usually used in conducting information systems (IS) research.

2.2 Strategies for Conducting Research

The general purpose of a strategy is to facilitate easier understanding and resolution of problems. Bryman (2004) describes a strategy as a “general orientation to the conduct of social research”. According to Olivier (2004), there are three principal strategies that can be used to conduct research. These are:

- a) Compiling information on a topic, in which the bits and pieces have already been discovered, (often by other researchers) but where the bits and pieces have not yet been integrated into a single coherent body of knowledge;
- b) Looking with new eyes at existing knowledge (standard ways of doing things) and trying to find a better solution for a problem that has previously been solved and, lastly,
- c) Solving a problem for which there is no known (or apparent) solution.

From the research interest and objectives of the research described in Chapter 1, this research is concerned with all three strategies. The purpose of the research is to integrate bits and pieces of knowledge that can be added to the software book of knowledge and the ontology research body of knowledge. It also searches for a solution to the problem of mechanistic development of software. The research strategies usually inform the researchers on the type of research they must undertake.

2.3 Types of Research in Information Systems

There are two major types of research: theoretical research and empirical research. These two types of research are governed by differing research problems and by the nature of these problems. The type of research implicitly or explicitly determines the type of theory

generated and is also guided by the research strategy adopted. The two types of research are discussed below.

2.3.1 Theoretical Research

Theoretical research is concerned with developing and refining a body of abstract understanding of phenomena and issues. Dahlbom and Mathiassen (1992) regard theory as something very fluid and possessing some romantic aspects. They regard theory development as a way of bringing order and sense to a complex real world. It is a process of structuring or finding the rules that govern the relationship between objects or artefacts that make up particular social phenomena. Bringing order and sense to a complex world is supported by Dahlbom (1996) as going away from the discernible phenomena to deeper, unseen layers of reality in an attempt to define concepts and general laws in terms of which the disorganized flux of visible facts can be systematized and explained. This notion deromanticizes the meaning of theory. Theoretical research may lead to empirical research.

2.3.2 Empirical Research

Empiricism relies on the notion that only knowledge gained through experience and senses is acceptable. It claims that, if ideas are revealed through grand- or middle-range theories, they should be subjected to the rigours of testing before the research fraternity can accept them as knowledge (Bryman, 2004). Empirical research requires the collection of data or facts relevant to the research problem. Empirical research is usually used to provide evidence to drive the process of theory development. In some cases, empirical data may emanate from a theory. The collection of facts is considered a legitimate goal in research and can justify the existence of knowledge. Some researchers describe this second process as naïve empiricism.

In the second stage, when data collection is complete, the data are critically analyzed to try and make sense of them (Cornford & Smithson, 1996). The major task in empirical research is to find inter-subjective observable knowledge among actors that justifies the existence of an occurrence. Processes such as the interview data-collection process used in this project are examples of the empirical issues referred to here. Regardless of whatever type of research is followed or one can follow, the researcher has to contend with three major types of research tasks.

2.4 Research Tasks in Information Systems

Both the theoretical and empirical types of research are associated with a research task, that is, the activities needed to accomplish a specific type of research. Both theoretical and

empirical researches can be conducted through constructive, nomothetic or ideographic research tasks. These three research tasks are discussed below.

2.4.1 Constructive Research Task

Constructive research is also known as inductive research. This research task is concerned with the development of frameworks, refinement of concepts or pursuance of technical developments (Cornford and Smithson, 1996). The task of the enquiry is to build theory as the main outcome (theory building). Observations are made and are subjected to scrutiny to check for trends and patterns in the data. The general relationships among data gathered are used to formulate a hypothesis or a proposition and, subsequently, a theory.

It may be necessary to validate the inductively generated theory by subjecting it to more tests using the deductive process. This process is commonly used in GTM. According to Iivari (1991), constructive research may develop models and frameworks that do not describe any existing reality. The purpose of these frameworks, therefore, is to create new forms of reality. If constructive research is used for theory building, researchers should then have a task that can be used to prove this theory.

2.4.2 Nomothetic Research Task

Also known as deductive research, nomothetic research is concerned with the exploration of empirical data. The idea is to use the data to test a theory or a hypothesis. Nomothetic research strives to find evidence to support general laws or theories that cover a whole class of cases (Cornford & Smithson, 1996). Loosely said, its task is to validate proposed theories empirically. The output of this task is deductive theory.

2.4.2.1 Deductive Theory

In all research, there has to be a link between theory and research. This link is established through the theoretical grounding provided using middle-range theories such as deductive theory. The deductive process starts by generating a hypothesis. This hypothesis is generated using existing knowledge about phenomena in a specific domain and the theoretical considerations in that domain (Bryman, 2004). The study of existing literature in the study discipline can be used to generate these hypotheses. This hypothesis is then subjected to rigorous empirical verification.

Literally, a hypothesis can be likened to a provisional idea that needs to be evaluated to find its merit. After evaluation a hypothesis may be accepted or rejected, depending on the

conformance of the data to the proposition. Bryman (2004) depicts a hypothesis as a carrier of concepts that need translation into researchable entities. The task of translating these concepts into researchable entities is termed operationalization.

During operationalization, the researcher specifies how data can be collected in relation to the concepts that make up the hypothesis (Bryman, 2004). The deductive process, therefore, starts with both theory and hypothesis. These later drive the process of data collection. The final reflections on the implications of the data gathered on the theory and hypothesis introduce some element of inductive process to the research. The third research task is called ideographic research.

2.4.3 Ideographic Research Task

Unlike constructive and nomothetic research tasks, an ideographic research task is concerned with exploring particular cases or events and providing the richest picture of what transpires. A phenomenon is taken individually and studied in its own right in order to understand its particular context. It emphasizes the analyses of subjective accounts based on particular or close association with everyday events. Ideographic research may lead to the generation of inductive theory. All these research tasks have to be guided by the way researchers conceive their world. What makes up the researcher's reality space? This can be explained using the sociological paradigms.

2.5 Sociological Paradigms Applied to Information Systems

A paradigm is defined by the Merriam-Webster online dictionary as a *philosophical or theoretical framework of any kind*. The framework encompasses assumptions, concepts values and practices that a community of people have when looking at reality. It is a way of seeing, thinking and interacting with phenomena in a reality space. In the software development field, Schach (2005:25) refers to a paradigm as a “model or a pattern” and not as a style of software development.

In line with this view, Burrell and Morgan (1979) developed four sociological paradigms that are now in widespread use in the research fraternity: the functionalist, interpretive, radical structuralist and the neo-humanist paradigms. These four paradigms, as used in information systems research, are depicted in Figure 2.1. Burrell and Morgan developed these paradigms as answers to four critical questions that always confront researchers when they look at the nature of the world.

The first question deals with the nature of reality. It questions whether researchers should accept reality as a given or not. In other words, does reality exist outside the observer's mind? On the other hand, should people conceive reality as a figment of their mind? There is a strong motivation for people to consider reality as a creation of the mind.

The second question deals with human understanding. Some researchers contend that understanding things require people to experience them. It is, therefore, important to know the determinants of human behaviour. The third dimension considers behaviour as a voluntary action. However, some argue that the environment can shape and determine the outcome of a person's behaviour. Lastly, the fourth question arises based on the factors that dictate human understanding: Should people use scientific methods in an attempt to gain understanding or should they directly experience the situation to understand it?

These four questions led the sociologists Burrell and Morgan (1979), to look at the nature of the world, using the ontological and epistemological axes shown in Figure 2.1 as a lens. Ontologically, the world can be considered as a continuum, ranging from a collection of ordered artefacts to a set of artefacts that are always in a state of conflict. Epistemologically, the world again can be considered as a continuum, viewed as consisting of objective artefacts through to subjective artefacts. The word epistemology, derived from the Greek word *episteme*, which means knowledge, looks at the forms of knowledge that can be obtained about the world. It is a concept that focuses on the truthfulness of knowledge relative to the observer of the reality. The four paradigms as used in information systems research are briefly discussed below.

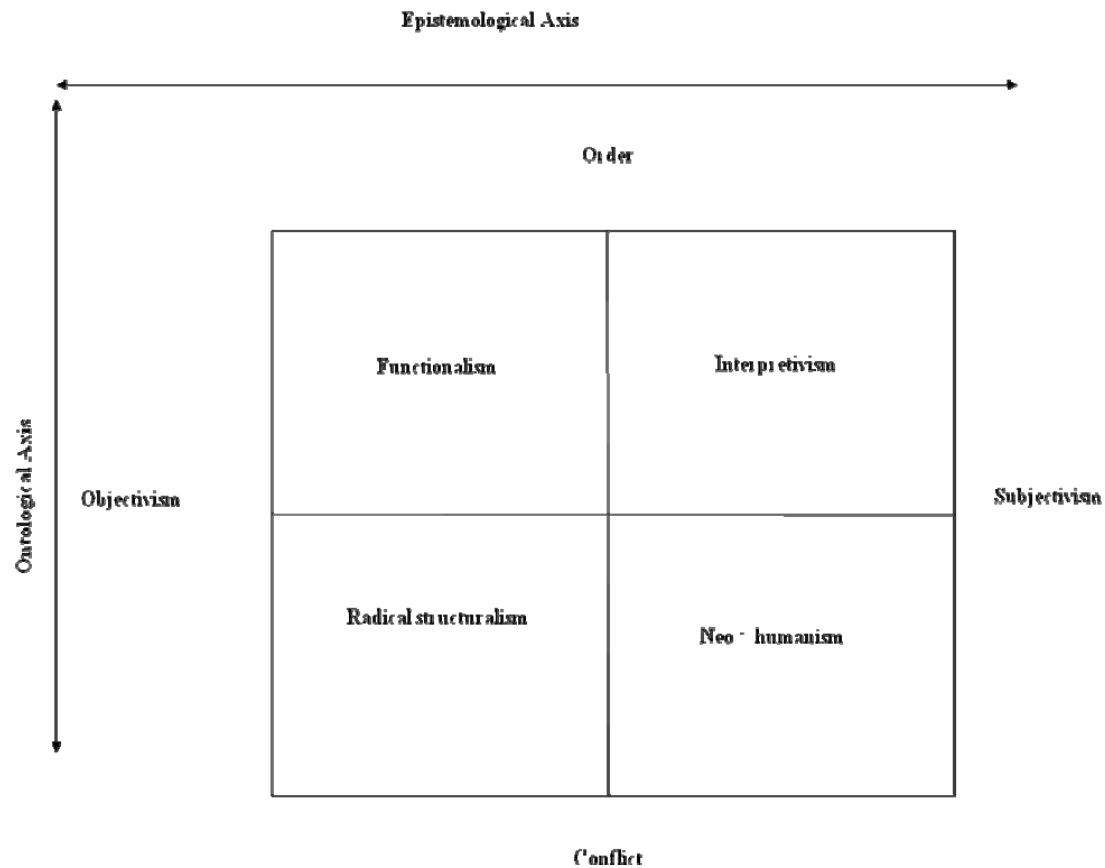


Figure 2.1: Sociological Paradigms (Adapted from Burrell and Morgan, 1979)

2.5.1 The Functionalist Paradigm

The functionalist paradigm perceives society as being made up of structures that can be aggregated to form a system. This system can be decomposed into its functional constituencies. The paradigm assumes that the social world is ordered and composed of relatively concrete empirical artefacts whose relationships can be identified, studied and measured through approaches derived from the natural sciences. Most information system problems have always been viewed and solved using this functionalist perspective. For more detail the reader is referred to the discussion on systematicity and system formation in Section 4.5.1.

Information systems professionals who work in this paradigm believe that the world, organizations and information systems are aggregations of functional units that are objective and ordered. This view allows system developers and researchers alike to use reductionist principles when solving information system problems. Using this approach, researchers assume a realist, positivist, determinist and nomothetic view to the world around them.

Functionalists accept the notion that general laws and theories applicable to the study of natural sciences can also be used to study human behaviour. In their quest to find truth, functionalists take the position that, by using the correct instruments and methods, the right procedures and value free investigation, one can objectively find the truth. They contend that the truth is “out there” waiting to be discovered. By the same norm, the researcher assumes the role of an objective and independent observer detached from the problem being researched. The researcher’s position as an independent observer is challenged by researchers working from the interpretive paradigm.

2.5.2 The Interpretive Paradigm

This paradigm is characterized by a need to understand the world as it is, to understand the fundamental nature of the social world at the level of subjective experience. It seeks explanations within the participant’s frame of reference, rather than from an objective observer of the action. Although it assumes an orderly world, it recognizes the existence of a subjective world, in which truth can only be found as a by-product of the observer. There is a greater need for researchers in this field to get to the level at which one can experience the phenomenon under study.

Interpretivists assume a nominalist, anti-positivist and voluntarist view of the world. At the methodological level, an ideographic stance is taken. Researchers perceive the world as an emergent social process, which is continuously created and recreated by people. The researcher becomes part of the researched and, only then, can one understand the world from within.

2.5.3 The Radical Structuralist Paradigm

It is important to note that radical structuralists share an objective departure point with the functionalists but, in addition assume that radical change is built into the very nature and structure of contemporary society. Radical structuralism focuses primarily on the structure and analysis of economic power relationships. It is assumed that society has inherent structural conflicts that result from political, economic and social forces, all of which aim to change the world.

In addition to sharing the same ontological view with the functionalist paradigm, the radical structuralist paradigm believes in society’s radical change, its emancipation and potentiality. Information systems researchers are more interested in the power relations that exist amongst actors in an organization and in how these power forces combine to effect radical change in

an organization. The world is viewed as being composed of artefacts that are always in a state of conflict.

2.5.4 The Neo-Humanist Paradigm

Unlike the radical structuralist paradigm that accepts the world as objective, the neo-humanist paradigm assumes a subjective interpretation of the social world, while accepting its conflicting and disorderly state. It differs from the interpretivist paradigm only in the sense that it is committed to overthrowing or transcending the limitations of the existing social world.

Information systems researchers working in this quadrant also share a romantic view of the world. This view of the world, while approximating the real world status quo, is challenged by the fact that not all the softer issues of society or world can be systematized, coded and subsequently implemented using technology.

Study of the four paradigms is of the utmost importance to information systems researchers so that they can understand the philosophical groundings of information systems and how people perceive the world. Once it is understood how people view the information system world, the ontological, epistemological and humanist nature of the information system problem to be solved has to be discussed.

2.6 Philosophical Groundings in Information Systems Research

When undertaking information system research, one should ground the research on some underlying philosophical assumptions. Questions on what constitutes valid research and what is the most appropriate research method should be answered (Myers, 1997). The underlying assumptions should be spelt out and identified both before and during the research process.

Also, in order to conduct and evaluate the research, it is important that the researcher identifies these sometimes hidden assumptions. Three philosophical assumptions: the ontological, epistemological and human nature, which are frequently used in the study of information systems, are discussed below.

2.6.1 Ontological Grounding to Information Systems Research

At the start of an investigation, the researcher should have an appreciation of what makes up the world. The researcher should find out what exists in the world and what can be known about this world. Also included in the search are people's conceptions of reality. The study of these underlying assumptions made about the phenomena under investigation is termed

ontology. As a field of study, Ontology (capital letter “O”) deals with the nature of social entities (Bryman, 2004).

The underlying assumptions in ontology exist as a continuum, ranging from nominalism through to realism. In between these two, materialism is found. Nominalism, usually referred to as idealism, is the belief that things are what we, as individuals, think they are. This nominalism positions the social structure of the world and its interpretation at the core of reality construction. The law of causality, cause and effect does not apply in the idealistic ontological position.

In the nominalist view, Roode (1993) advocates the inclusion of structures that are created by individuals through the processes of naming, labelling, and defining concepts. He argues that there are no invariant structures in the world that are waiting to be discovered “out there”. In creating reality, everything needs the consciousness of individual human beings. Meaning given to artefacts is created and recreated through interactions between people, technology and the environment in which they all reside.

In contrast to the nominalist assumption, realism assumes that the world is made up of objective facets that possess concrete characteristics. It holds that there is an external reality, detached from the observer’s beliefs and perceptions (Reetley, 2003).

Lying between nominalism and realism, materialism is found as a notion that argues that reality exists but that this reality is only confined to material features such as physical objects in the world (Reetley, 2003). Values, beliefs and experiences are not considered as contributing to the existence of reality.

2.6.1.1 Objectivism

Objectivism deals with the question of whether social entities have an objective reality that is external to social actors or not. In fact, it addresses the issue of whether reality, human conceptions and their interpretations exist independently (Reetley, 2003).

Objectivism views social phenomena and their meaning as external facts whose structure and existence cannot be influenced by the observer. This notion accepts that there exists a reality that is external to people who act or interact with social phenomena. Reality is viewed as being independent of the social actors that work or act on the phenomena and that there is a reality somewhere “out there” that is always waiting to be discovered.

2.6.1.2 Constructionism (Constructivism)

Constructivism on the other hand, discusses whether social activities are the result of some social construction that evolves in an actor-network environment as a result of the perceptions and actions of social actors (Bryman, 2004). This views social phenomena as having emergent properties. Reality is always in a continuous state of construction and reconstruction. Social interaction creates social phenomena and the classes that are thus produced are in a continuous state of revision. The way meaning is assigned a contextual attribute is a typical example of this. In short, social objects and categories are socially constructed. Data can be processed into information. People can deduce the meaning of social artefacts within a context. The sharing or existence of a commonly accepted inter-subjective context leads to general meaning being given to social artefacts. In addition, context is time- and place-dependent. As contextual meaning leads to knowledge, knowledge is a social construction. Constructionism can be used to reflect the inadequacy of our knowledge of the social world. There is thus a continuous creation and re-creation of knowledge in the world.

2.6.2 Epistemological Grounding to Information Systems Research

Cornford and Smithson (1996) describe epistemology as a type of (valid) knowledge that can be obtained about a phenomenon being studied. Olivier (2004) refers to this as the knowledge that people can and cannot have. Olivier's argument is that some people may possess knowledge unknowingly since it is the expertise and skills acquired by a person through experience or education. Epistemology can also be regarded as the study of how organisms get to think, decide and come to know their social world (Reetley, 2003). In other words, it is the study of what can be regarded as acceptable knowledge in a specific discipline. However, this study is not a static discipline, as Hacking (2002:8) explains:

"...epistemological concepts are not constants, free standing ideas that are just there, timelessly."

These concepts, as knowledge, are mutable, as several successive generations of researchers have used them to organize knowledge and to conduct scientific inquiries.

As depicted in Figure 2.1, the epistemological axis runs as a continuum from objectivism (positivism) through to subjectivism (anti-positivism). This continuum covers the positivist notions of the natural sciences and the realist notion (which is usually called critical realism) through to interpretivism. These epistemological notions are discussed below.

2.6.2.1 Positivism versus Anti-Positivism

The positivist strand believes in realism, that is, that there exists a single view of the real world that we all share. Any observer of this real world will get the same view and results, which will be independent of the observer (observer-free perspective).

In the objective world, the social world can be subjected to the same form of enquiry as the natural world, which consists of models and methods. As human action and institutional aspects inherent in the social systems are totally independent of each other, we can thus have an observer-independent result from such an investigation (Cornford & Smithson, 1996; Roode, 2004).

On the other hand, the anti-positivist notion believes in relativism. Relativism accepts that there is no real world “out there” waiting to be discovered. Everything occurs as a result of constructions in individual human minds. Culture, society, language and the interaction of these will help to shape what is observed by researchers during their investigations. Issues such as time and place begin to make impact in the results of observations in a relativistic environment. Ideologically, anti-positivists believe that all knowledge, both scientific and non-scientific, is a result of social construction. Knowledge is formed and shaped by the values of a society of interacting actors. This school of thought holds that there is no clear distinction between facts and values (Cornford & Smithson, 1996).

With regard to information systems, there is no hard, real and already tangible information or knowledge that can be found. The knowledge is of a softer, more subjective, spiritual or even transcendental form that is based on the experiences and insights of the unique and essentially personal nature of the actors (Roode, 1993). This results in many problems in this field requiring a subjectivist approach to their solutions.

To summarize, positivists regard all these explanations as being based on regularity and causal relationships, whereas anti-positivists only give validity of knowledge to the viewpoint of the participant within a given activity. Anti-positivists contend that knowledge progresses from an objective to a subjective reality.

2.6.3 The Humanistic Grounding in Information Systems Research

This philosophical assumption considers the relationship between human beings (actors) and their environment. The experiences of actors in information systems are not necessarily prescribed products of the environment in which they exist. Their actions are not

deterministic, although many developers using the functionalist paradigm approach regard them as deterministic. Determinism leads to the bias of many developers who assume immutable functional models instead of mutable action models. Information system actors, however, create their own experiences through interactions with the environment. Roode (1993) described this notion as the voluntarist view.

This humanistic-culturalist tradition takes as its epistemology the fact that, in a socio-technical environment, knowledge is constantly being created through the interaction of actors. According to Checkland (1999:277), actors are always “negotiating their interpretations of reality, those multiple interpretations at the same time constituting the reality itself”. This is supported by Giddens (1976) who argues that knowledge is not a pre-given entity but one that is continuously created by the actors in a social setting and manifests itself as a shared cognitive base of the participating actors. The inclusion or existence of people as part of information systems makes these systems non-deterministic, unpredictable and, therefore, socially constructed systems.

These three philosophical groundings are used to inform the researcher about the correct research paradigm to use in this study. In addition to the sociological paradigms, they guide the researcher in his choice of the correct pair of world views, i.e. objective-order, objective-conflict, subjective-order or subjective-conflict pairs to position the nature of the research problem.

2.7 Information Systems Research Paradigms

There are basically two strands that a research project can follow. These strands (referred to here as research paradigms) are the qualitative and quantitative paradigms. The research paradigms can be likened to a strategy, as a way of thinking or as a way of looking at things. The research paradigm to be followed is dependent on the ontological and epistemological positions taken by the researcher about the problem. These two paradigms are described in Sections 2.7.1 and 2.7.2 below.

2.7.1 Quantitative Research Paradigm

This paradigm is usually used by researchers operating in the functionalist paradigm. Quantitative research is based on the assumption that natural phenomena can be quantified in numeric terms. The development of metrics (numbers) that can be used to describe phenomena (objects and relationships) under study are the focus of this type of research

(Cornford & Smithson, 1996). In addition to quantification in the collection and analysis of data, quantitative approaches entail the following (Bryman, 2004):

- They follow a deductive approach to the relationship between theory and research putting emphasis on the testing of theories.
- They incorporate the practices and norms of the natural scientific model. Hence they follow the positivist epistemological stance.
- Ontologically, they consider social reality as an external objective reality.

2.7.2 Qualitative Research Paradigm

Qualitative research targets the production of theories. Cornford and Smithson (1996) describe qualitative research as one that avoids metrication and search for other means of capturing and analyzing (understanding) data. Qualitative research involves the use of qualitative data, such as interviews, documents, and participant observation data, to enable social phenomena to be explained and understood (Myers, 1997). In qualitative research, as Reetley (2003) explains, verification and extension of theory are not the primary purposes of data collection. The intention is not to predict causes of human behaviour, but to understand how humans derive sense and meaning from their day-to-day actions. According to Olivier (2004), in qualitative research, the data to be used in a research, their appropriateness and their processing are issues that are all decided by the researcher.

Myers (1997) identifies peoples' ability to talk as the distinction between people and the natural world. He used this distinction to motivate researchers to embrace qualitative research methods. Qualitative research methods are designed to help researchers to understand not only people, but the social and cultural contexts in which they live. Research in this paradigm has been associated with the interpretivist and relativist positions as expounded from the epistemological anti-positivist view of the real world.

The qualitative paradigm negates the generalizability which is inherent in models and theories of the natural sciences. However this qualitative paradigm does not imply the absence of numbers in the research. It focuses on how the analyses which led to the conclusion or findings of the research were executed. Cornford and Smithson (1996) contend that qualitative research supports the interpretation of the results and thus supports the nature of reality. Nature has to be interpreted and understood.

When conducting qualitative research, it is very difficult to use a highly systematic research approach. A systematic approach may lead to the loss of information that is potentially available in qualitative research data. Bryman (2004) summarizes the basic tenets of qualitative approach as follows:

- It relies on descriptions of phenomena during the collection and analysis of data.
- It generally uses an inductive approach to the relationship between theory and research and emphasizes the generation of theories.
- It relies solely on the way in which people interpret their social world (interpretivism).
- It views social reality as a constantly shifting emergent property of individuals' creation.

Qualitative data analysis is, therefore, a procedure that seeks to produce findings, concepts or hypotheses without the use of statistical methods (Glaser, 1992). The researcher believes that, after choosing the correct research paradigm, one should focus on the choice of the right research approach.

2.8 Approaches to Information Systems Research

There are two basic approaches to information systems research that are governed by whether one is a positivist or an anti-positivist. The research problem space, like the reality space, cannot be treated as discrete but as a continuum. Some research problems, therefore, lie along the line joining the positivist and the anti-positivist poles. Problems found along this line should, therefore, blend the positivist approaches and the interpretive approaches that are usually used by the anti-positivists. These research problems are, therefore, dealt with using the critical realist approach. The three research approaches: positivism, interpretivism and critical realism, may be regarded as the theoretical perspectives or frameworks to research (Sarantakos, 1997; Henning, 2004). More often than not, these are referred to as methodological groundings of the research.

2.8.1 The Positivist Stance

Bryman (2004) describes the positivist approach as a philosophical position that relies on crude and superficial data collection during the research process. In addition, as Reetley (2003) puts it, positivism calls for a study that is separated from the subjective elements of experience. Positivist research treats the subject of research as a subject capable of

developing general theories of universal applicability (Cornford & Smithson, 1996). In some cases the theories need to be proposed and tested.

The primary goal is to limit the impact of the researcher on the results. Cornford and Smithson (1996:27) added that positivist research regards true knowledge from research as a result of an “observation or experience of real phenomena in an objective and real world”. The end product is regarded as undisputed facts that are timeless and without any social values attached. The researcher does not influence the results of an investigation and, assuming that all variables are kept constant, other researchers may reach the same conclusion as the first researcher, regardless of differences in personal traits (Olivier, 2004).

Bryman (2004) noted the following as the most accepted notions in positivism and the author states definitely that:

- Only phenomena and, hence, knowledge confirmed by the senses can genuinely be warranted as knowledge. This is called the principle of phenomenalism.
- The purpose of theory is to generate hypotheses that can be tested. These hypotheses will thereby provide explanation of laws to be assessed. This is called the principle of deductivism.
- Knowledge can only be found through a fact-gathering process. This provides the basis for generating laws and is called the principle of inductivism.
- Science can only be conducted in a value-free way. This is called the principle of objectivism.
- Finally, scientific statements differ from normative statements. The true scientist belongs in the scientific domain. This supports the notion of real knowledge being gathered and confirmed by the senses. Normative statements cannot, however, be confirmed by the senses.

Positivists believe that the scientist’s conceptualization of reality directly reflects that reality and that the concern should be to find explanations of human behaviour.

2.8.2 The Interpretive Stance

Interpretivism is a school of thought that emphasizes the importance of interpretation and observation in attempts to understand the social world (Reetley, 2003). It shares the view that the subject of enquiry in the social sciences, that is, people and their institutions, is

fundamentally different from that in the natural sciences and is in fact an integral part of the qualitative research paradigm. The epistemological grounding of interpretivism lies in the understanding of human behaviour that is arrived at through a series of cyclic processes of interpretation that are referred to as the hermeneutic cycle. The emphasis is on understanding human action and not on the forces acting on it.

Interpretive qualitative research uses the understanding of what is being researched as its primary goal. Avison *et al.*, (2008:11-12) classify it as research that is used in predicting and explaining “the status quo”. The prime consideration is the fact that understanding is subjective and depends on the culture, language, history and background of the researcher (Olivier, 2004). Since language is socially constituted, the interaction between the researcher and participants will impact on the results. Furthermore, since language is subjective, all the interpretations arrived at, therefore, depict the researcher’s or observer’s point of view.

Interpretive researchers start with the assumption that access to reality (whether given or socially constructed) is only obtained through social constructions such as language, consciousness and shared meanings (Myers, 1997). Interpretive studies generally attempt to understand phenomena through the meanings that people assign to them. In addition to the hermeneutic–phenomenological tradition usually associated with interpretivism, Max Weber’s (1864-1920) notion of *Verstehen* (understanding) and Mead’s symbolic interactionism (Bryman, 2004) are included in the epistemological set of interpretivism.

Walsham (1993) regards interpretive methods, when applied to information systems research, as a task of understanding the dual and reciprocal effect of context on information systems and vice versa. Interpretive research does not predefine dependent and independent variables, but focuses on the full complexity of human sense-making as the situation emerges (Kaplan & Maxwell, 1994). The interpretivist’s world is concerned with the examination of the “life world”: the world of consciousness and of humanly created meanings. As Avison *et al.*, (2008) put it, interpretive research is used in predicting and explaining the status quo.

Interpretive research is thus regarded as being very fluid. Its fluidity and the grounding upon which it is based support the use in this research of the romanticized definition of theory.

2.8.3 The Critical Realism Stance

This lies between the positivist and interpretive stances. This realism holds that the natural world and the social sciences can and should apply the same kinds of approach to data collection and to its explanation. It can be split into empirical realism and critical realism.

Empirical realism, sometimes referred to as naïve realism, assumes a perfect correspondence between reality and the terms used to describe it (Bryman, 2004). This type of realism also holds that, with the use of appropriate methods, reality can be clearly discerned or understood.

On the other hand, critical realism believes in an external reality that exists and is separate from the observer's description of it. It contends that the social world can be understood and so changed if we manage to identify the structures at play that generate events and discourses in the social world. The structures are, however, a result of practical and theoretical work in the social sciences (Bhaskar, 1989).

Therefore, critical research, which is based on the grounding of critical realism, strikes a balance between positivist qualitative research and interpretive qualitative research. It is grounded on the fact that any artefact observed by a researcher in society is socially constructed. According to Mingers (2002), the critical research approach assumes the existence of a domain of structures and mechanisms, events and experiences (the Real). These structures may be physical, social, or conceptual and may well be unobservable except through their effects. It also recognizes that our knowledge is always provisional and is historically and culturally relative and that people do not have observer-independent access to the world. Critical research is about evaluation and transformation of the social reality that is the subject of investigation (Avison *et al.*, 2008).

Critical realism argues that scientists' conceptualizations of reality are just one way of trying to understand or know that reality. There is a distinction between the objects that are the focus of any enquiry and the terms or concepts that are used to describe those objects. Although generative mechanisms may not be observable, they are acceptable as long as their effects are observable (Bryman, 2004). These discussions on research approaches led the researcher to focus on the nature of the research problem in this study.

2.9 The Nature of the Research Problem

The discussion on the philosophical underpinnings of information systems research provides a lens through which the nature of the research problem can be identified. In research, three basic concerns about the problem have to be addressed: the ontological stance, the epistemological stance and the humanist stance.

A clear diagnosis of the nature of the information systems problem along these three basic assumptions is important, since this will be used to inform the methodological design of the research.

2.9.1 Ontological Diagnosis

Organizations and their information systems are considered as vast actor-networks. Although much of the technological components of these networks can be structurally separated, unfortunately their effects in the whole system cannot. In real terms, the socio-technical nature of the problem requires the neo-humanist stance to be adopted but there are also limitations on what can be observed, interpreted and implemented in software development. What really exists should be regarded holistically from the interpretive paradigm, anti-positivist stance because of the limitations caused by the formalization requirements in all system implementations.

2.9.2 Epistemological Diagnosis

The knowledge that people have about the world, especially about organizations and information systems is always provisional and in a continuous state of creation and recreation. Using the actor/network theory and structuration theory, coupled with the fact that meaning and knowledge have a situated practice that is heavily dependent on the context, the research should adopt a relativistic and voluntarist stance.

2.9.3 Humanist Diagnosis

Many of the problems facing software development and system development emanate from the fact that developers consider organizational problems as deterministic and rational. In view of this, hard software development methodologies have been used, an issue that has created many problems for the software developer. When solving software development problems, researchers are urged to take the voluntarist stance. This supports the fact that humans create their own environment and are neither shaped nor modelled by it. Although this stance has limitations when it comes to implementation of software products and systems, it will be considered continuously throughout this study.

However, in the implementation of any system, the world has to be considered as being structurally ordered to some extent. Hence we subject the world to some formalisms and conceptualizations which have to fit some defined intention.

In conclusion, the research rejects the objectivist position and calls for the subjectivist acceptance of how people view and interpret reality. This research is concerned with the creation, modification and interpretation of the world. This posits the research as relativistic throughout the project.

2.9.4 Methodological Stance

In any research, understanding the nature of that which is to be studied is of paramount importance. From the above it can be seen that research can be either quantitative or qualitative. The major goal of this research is to find ways of improving the development of software products, which are later incorporated into information systems. By its nature, the problem under investigation cannot truly yield purely objective results, irrespective of the research approach used in the study. This process makes the present research interpretive in nature. The research method to be adopted for this study “must be appropriate for the research question” (Avison *et al.*, 2008:11). In addition, it should be applied appropriately relative to the research problem under investigation. Suddaby (2006) concurs, saying that there should always be consistency between the research questions, the assumptions made about the world view and how people perceive it, and the methods used to address these research questions. This chapter has attempted to find a link between these tenets.

2.10 Summary

The contents of this chapter can be summarized as the '*Philosophy of Information Systems Research*'. Of importance to its inclusion in this thesis is the fact that it discussed most of the basic philosophical groundings in information systems. This chapter is a blueprint on thinking about research in the information systems discipline. It provides a road map to issues that should be discussed and embraced by researchers whenever they decide to conduct research into information systems. Researchers should understand that information systems problems vary in structure, form and context. This alone requires people to understand the philosophical nature of these problems before selecting a research methodology.

Any type of information system research should, however, answer the following questions:

- What type of research is being undertaken? This looks at the dichotomy between theoretical and empirical types of research.
- What is the purpose of the research: generation of theory or testing of theory? The research task should then be constructive, nomothetic or ideographic.

- With regard to the research problem, how should nature be conceived? The four sociological paradigms, together with the ontological, epistemological and humanist philosophical underpinnings, should be used as lenses to understand the research problem.
- What research paradigm should an information systems research project follow? This question is answered by the dichotomy between qualitative and quantitative research paradigms in conjunction with the three approaches to information system research: positivist, interpretive and critical realism.
- The last question then addresses the elicitation or positioning of the nature of the research problem. This part should guide the researcher in the selection of the methodological process for the research.

In conclusion, this research is equally theoretical and partly empirical, constructive and partly ideographic. It assumes a relativist, anti-positivist, interpretive stance and will use GTM as the research method. In Chapter 3, *Research Methodology and Design*, the methodological processes followed in this study are discussed.

CHAPTER THREE

RESEARCH METHODOLOGY AND DESIGN

3.1 Introduction

Chapter 2 discussed the philosophical underpinnings of IS research. These will be used to guide the research in picking an appropriate research methodology and in Chapter 4 they will guide the researcher in motivating for a correct theoretical conceptual framework that guide the investigation of the phenomenon under study.

“...adequacy of a theory ... cannot be divorced from the process by which it is generated.”

Barney G. Glaser and Anselm L. Strauss, 1967.

Every research process, whether deductive or inductive, has to use some theory. Glaser and Strauss (1967) emphasized the need to chose, plan, execute carefully and document comprehensively all the processes that are undertaken during research if the research goal is to generate theory. This task contributes immensely to the acceptance and final use of the theory in practice.

When researching phenomena that do not fall into the functionalist and positivistic paradigms, researchers find it more compelling to spend some time in choosing the most appropriate research methods. Most importantly, if the research is qualitative and interpretive in nature, more emphasis should be placed on the research design process in an attempt to ensure that two factors, quality and rigour, are satisfied. This chapter discusses the methodological facets needed to come up with good research whose output can be relied upon.

Firstly, the philosophical nature of a research problem dictates the research approach and methodology that will be followed. To borrow Glaser’s dictum (1992), a methodology can be described as a theory or study of methods. A single research project may use one or several methods. It is, therefore, essential to order and describe these methods in a coherent way. We begin this chapter by discussing the philosophical assumptions made about the research problem. We then discuss the research approach. A brief discussion on qualitative and interpretive research then follows. In order to familiarize the reader with various qualitative research methods, a very brief discussion of these methods is given in this chapter.

In Sections 3.3 to 3.6, the GTM, the research method used in this study, is discussed in detail. This discussion is in conjunction with the discussion on research design and, lastly, the

quality requirements of an interpretive research are outlined. The chapter concludes with an analysis of the criteria for ensuring quality and rigour in qualitative information systems research as used in this research.

3.2 Research and Methodological Approach

This section is devoted to the discussion of the research and of methodological approach used in this study. It highlights the relationship between sound theory generation and the processes involved in its development. It is of the utmost importance to explicitly explain the meanings given to research approach, methodology and method, as opposed to the use of these terms in the software development field.

3.2.1 Research Approach

In a research approach we look at the pragmatic question of how the step-by-step execution of a research project is carried out. An approach may be defined as an overall idea of how the project will be undertaken. Depending on the degree of complexity of the problem, a single project may require one or a triangulation of techniques from both the qualitative and quantitative research paradigms. It may also require the use of both the inductive and deductive techniques as expounded by Bryman (2004). Some researches move from inductive reasoning to theory generation and then to deductive testing of the theory for validity, using some empirical methods. Others may just require the empirical testing of a theory, that is, the deductive process only. This research is an inductive type of study and does not require the testing of the generated theory. This study, however, calls upon the use of research techniques from the qualitative paradigm only.

3.2.2 Methodological Approach

The methodological approach consists of the research methods, research design, data-acquisition methods, modes of data analysis employed in the research and the justifications for employing all these techniques. In this regard, research methodologies are viewed as translation languages that map the principles of a paradigm to actionable steps to be followed in a research project, that is, the methods. This research assumes the definition of a methodology, the science of methods, to be a set of research principles that are related very closely to a unique paradigm. The methodology as a translation language provides “guidelines on acceptable research practices” (Sarantakos, 1997:34). This section discusses each of these methodological components in general and then in relation to how they have

been applied to this study.

3.2.2.1 Qualitative Research Methods

A research method may be seen as a “general way of thinking about conducting qualitative research” (Trochim, n.d). These methods require the purpose of the research, the part played by the investigator and the research phases (including the data-gathering and data-analysis techniques) to be described either “explicitly or implicitly”. Research methods are more prescriptive and are dictated by the research methodology chosen. Different methods can be used in different methodologies but the reverse is not always true. Sarantakos (1997:34) sums it up by saying “methods are a-theoretical and a-methodological”. The choice of research method influences the way in which the researcher collects and analyses data.

We will start by discussing some of the methods used in qualitative research. Finally, we will discuss the method used in this study. The discussion, however, is not exhaustive but only serves to give the reader an appreciation of why the researcher used GTM in this study. For a more comprehensive understanding of some of these different research methods in information systems and qualitative research, the reader is referred to Goede (2005:31-37) and Alexander (2002:30-38) who took the most trouble to discuss the pros and cons of a variety of these methods in their theses. Also, Myers (2003) has dedicated a website hosted by MIS Quarterly and the University of Auckland, to the discussion of most of these methods.

Ethnography

This method originated as a derivative of two Greek words namely, *ethnos*, which refers to people or ethnicity and *graphein*, which refers to the art of writing (Wikipedia, n.d.) or to a geographic location (Trochim, n.d.). Its philosophy is based on the assumption that, when one is studying culture, one needs to immerse oneself in the society whose culture is the subject of the study. In short, ethnography is the study of a social entity or practice in its geographical location.

The study of culture can only be done in conjunction with the people or society in which the culture resides. They cannot be separated. In modern day research, one cannot separate the culture of an organization from the organization itself. When applied to information systems, one cannot separate the system under study from the organizational context or from the people who will eventually use the system.

In such a case, the developer needs to be part of the system under study and also be able to capture the context, the culture and the political aspects inherent in the organization. The researcher assumes the role of an active participant in addition to being an observer of the other actors in the system.

Hunter (2003:297) qualifies this study as an exploratory type of study that has to be done over an “extended period of time”.

Phenomenology

This method looks at the subjective experiences and interpretations of people who are the subjects of the research. The way others view the world is the focus of the study. As described in SEP (2008), it is “the study of structures of consciousness as experienced from the first-person point of view.” In other words, it is a study of how people experience the presence of artefacts around them.

Field Research

This method can double as both a qualitative research method and a data-gathering technique (Trochim, n.d.; Hunter, 2003). As a method it grew out of its frequent and general use as a data-gathering technique. Researchers could not separate it from a method; hence its inception and cooption as a method in its own right.

Using this method, the researcher goes into the field and collects as much field data as is possible about the study area. Data may be collected by means of interviews or questionnaires. These are later coded and analyzed.

Field studies can be used in both positivist and interpretive researches. In the interpretivist tradition, the researcher does not treat data collected as objective facts but as products of individual respondents’ interpretations of the situation.

Action Research

The purpose of action research is to test a theory or a hypothesis. In this particular context the researcher is an active participant who engages with other participants (respondents) working in the area under study. The idea is to allow the researcher to be a learner as well as an inquirer in his/her extended relationship with the problem (Hunter, 2003:295).

The action researcher participates in identifying the problem, finding the solution and implementing the solution if needed, in the organization in which the research is carried out. The action researcher goes through a continuous, cyclic process of arriving at a solution,

getting feedback on the appropriateness of the intervention and fine-tuning the solution through a process of further data collection and analysis until the problem is solved.

Narrative Inquiry

Narrative inquiry is a research method in which participants are allowed to relate stories about events that have occurred. The researcher's task is to evaluate and rate the stories as being as correct as is possible. Through the participants' accounts of the events that occurred, the relationships between data in the area of study can be revealed (Hunter 2003:299).

Case Study

Myers (2003) notes that a case study can serve both as a unit of analysis as well as a research method. This partly supports Trochim (n.d.) who does not include it in his discussion on methods but regards it as a data-gathering technique. In contrast to this view, Hunter (2003) and Myers (2003) classify it as a research method. As a research method, a case study can be used in both the quantitative and qualitative paradigms. The main purpose is not to generate general laws but to understand phenomena in their context. The methods discussed here can all be used in conducting qualitative–interpretive research studies. However, this study used GTM which is discussed in the following section.

3.3 Research Method Used in this Study

This research followed a GTM study approach. There are many varied ways of conducting research using GTM, all grounded on either one of the two basic GTM strands, the Glaserian or the Straussian strand. These strands however, always leave researchers wondering which one of the two is better than the other and also looking for their similarities and differences in practice (Van Niekerk & Roode, 2009). However, there is a great difference in how these two strands are used in research as noted by Matavire and Brown (2008). Although some of these are very prescriptive (Strauss & Corbin, 1990), others give the researcher scope to direct the research in a way that suits the research environment. The proponents of GTM (Glaser & Strauss, 1967) and also supported by Charmaz (2006:9), however, urge researchers to use the method flexibly. Charmaz (2006) thus refuses to accept any prescriptive way of using this method. Instead she regards the method as a guiding framework that is as, “a set of principles and practices” that any researcher can fine-tune to suit the context of the particular research project (Charmaz, 2006:9). This argument was adopted in this study, where most of the original dicta have been modified to suit the dictates of the research environment and the phenomenon under study.

3.3.1 GTM

GTM is a research method that seeks to develop theory that is grounded in data. The main idea of GTM is supported by Hacking (2002:15), who, in his discussion of on the “Creation of Phenomena”, asked the question “What comes first, theory or experiment?” Hall, the discoverer of the Hall Effect in magnetism, however, insisted that experimentation should be the beginning of a theory and that, for anything to exist, it should be created.

Trochim (n.d.) regards GTM as a generative method whose purpose is to generate or produce theory. Although GTM can produce new knowledge, research findings require a constant and sustained process of “explaining a multitude of factors” that also raise a need for further elaboration (Trim & Lee, 2004:478). The grounded theory approach can be used to produce developing theory, that is, new theory that has not fully reached saturation point. This theory may not necessarily equate to the substantive theory as described by Glaser and Strauss (1967) and others. Glaser and Strauss (1967) urge grounded theorists not to start with a problem statement or research questions, saying that merely an interest in the field will suffice. This argument was, however, refuted in later studies as discussed in Strauss and Corbin (1990), Charmaz (2006) and Leedy and Ormrod (2000). However, adherence to this requirement influences the way the grounded theory researcher plans and executes the research study, as will be described below.

3.3.2 The Research Design

Trochim (n.d.) regards research design as a process or phase that binds the project components together. Each research activity is positioned and described according to its contribution to the overall goal of the research. It is a process of moving from methodological abstractions to descriptions of the practical steps that are proposed and later followed during the execution of the research. Since our chosen research method is GTM, the research design should fit the requirements needed for its use in this research. This section is dedicated to the discussion of this research design and its suitability for the interpretive type of research.

3.3.2.1 Requirements for Sound Application of GTM

In a grounded theory research project, a researcher with an interest in a specific area of study should first identify its substantive area. It is in this area that the developed theory should be applicable and used. In some instances, the area of interest may straddle several disciplines, leaving the researcher with the problem of explicitly defining the substantive area of research.

An example of a research study that straddles two substantive areas is given by Goede (2005). Her research focused on the use of data-warehousing in decision-support systems and system-thinking methodologies. Analysis of this research may mistakenly lead one to identify either system-thinking methodologies or data-warehousing as the substantive area in the decision-support system (Goede, 2005:59-66). This could completely change the research strategy of the study. In other circumstances, both may be selected as substantive areas of research. A critical look at this research, as explained by Goede (2008), reveals that she:

“... wanted to study one area from the viewpoint of the other area. I wanted to look at Data Warehousing (DW) from a Soft System Thinking (SST) perspective and not from an objective perspective - I did not want DW answers, I wanted DW from a SST perspective answers”.

In other words, the substantive area was an integral collection of data-warehousing practices viewed from SST as a lens and not separately.

It requires a very critical mind to successfully identify the focus point for literature study and substantive area. Such a multiplicity of disciplines which need to be consulted in the same research at one point or other may start to flout many of the basic dicta of GTM as proposed by Glaser and Strauss (1967).

In this research, there are also two fundamental disciplines that need to be consulted by the researcher. These are the fields of software development and ontology as used in information systems discipline. This dialectical debate emanates from the researcher's interest in using ontology to improve the development methods of information systems. This alone forces the researcher to shift his interest into the way these information systems are developed and, later, into the software products that are used to implement these systems.

The substantive area debate

This study recognizes that the title of the research project, “Towards an ontology-driven software development approach: An unended quest” is multifaceted. It is, therefore, very important that we highlight here that the substantive area is neither software development nor the study of ontology in information systems, but is an integration of both of these. Put more simply, it is the study and use of ontology in the development of software products.

As such, the study does not need to separate software development and the ontology discipline and later choose either one of them as its substantive area. It, therefore, treats the

two fields as substantive areas and any dictum applying to the sound use of GTM principles will be applied with respect to this resolution.

3.3.2.2 Problem Statement and Use of Research Questions

As part of the research design, Glaser and Strauss (1967) and Glaser (1992) call on grounded theory researchers not to conceive a research question or hypothesis as a statement that focuses on and identifies the unit under study. Many research methodologies call for a research hypothesis into the problem area that guides the method of data collection and identifies the unit of analysis. In contrast, using GTM, the research focus becomes clear during open coding, collection of data by theoretical sampling and analysis of the data through the constant comparison method. Since the researcher does not start with a problem statement but with an interest in the field, he cannot derive these research questions (Glaser & Strauss, 1967; Glaser, 1992).

In contrast to the suggestion made by Glaser and Strauss (1967) and Glaser (1992), Strauss and Corbin (1990:37) advise researchers to start with a preliminary theory, research question or hypothesis. This theory is intended to guide the researcher and define the scope of the study. They argue that, without such a research question, the researcher will be faced with too many aspects to consider in a single research study. However, in choosing such a research question, the researcher should structure this in such a way that it leaves flexibility and freedom for an in-depth exploration. It should not limit the investigator but should only be a guide (Strauss & Corbin, 1990:37). As the research progresses, the research question is narrowed and focused using concepts and relationships to concepts that are inductively arrived at after analysis of the initial data samples.

Strauss and Corbin's (1990) suggestion did not go down well with Glaser (1992), who accused them of encouraging researchers to limit the free generation of theory by introducing preconceived ideas. However, in her book, "Constructing Grounded Theory", Charmaz (2006) concurs with Strauss and Corbin, as she also advocates the use of a preliminary theory.

Despite this academic debate, in practice researchers have to navigate through all of these requirements and come up with a research design that is practical. The present author would like to negate Glaser's (1992) inflexible and strict adherence to the non-use of preliminary theory. After all, study of literature has shown that he does not allow people to criticise his

dicta. This is supported by some of the arguments reflected in his writings such as Glaser (1992), Glaser (1993) and Glaser (2002), addressing his discomfort with Charmaz's (2000) monograph, which suggested that grounded theory be viewed as a constructivist method.

In this research, the author took the risk of provoking Glaser's wrath by proposing preliminary propositions as well as research questions (*Chapter 1, Sections 1.4 and 1.5*). These, however, were strictly preliminary, as advocated by Strauss and Corbin (1990), Gasson (2003) and Charmaz (2006). This is also supported with a study done by Matavire and Brown (2008:145), where 11% of the articles that followed the so-called Straussian strand had a priori theory. In contrast, those that followed the Glaserian GTM strand avoided a priori theory altogether.

3.3.2.3 Use of Literature

"There is a need not to review any of the literature in the substantive area under study."

Barney Glaser, 1992:31

When doing quantitative studies, literature can be used to find relevant previous research in the area, to discover gaps in the knowledge and to find theoretical or conceptual frameworks that are used to guide the research process. When doing qualitative studies, in particular when using GTM, researchers are strongly advised to defer the literature study until such time as they have collected and analyzed the first batch of the research data (Glaser & Strauss, 1967; Glaser 1992 & 1993; Suddaby, 2006).

In using GTM, the intention is to discover concepts and hypotheses and not to test or duplicate them. At the beginning of the study, the researcher is advised not to study any literature in the field of study. This, Glaser warns, will introduce researcher bias by having a set of preconceived concepts, categories and properties from other researchers' work. Starting with a literature study will constrain the free discovery of theory and hence defeat the main dictum of grounded theory approach (Glaser & Strauss, 1967; Strauss & Corbin, 1990; Glaser, 1992; Charmaz, 2006; Gasson, 2003). This notion is supported by Hunter (2000:33), who argues that approaching a research problem without preconceptions leads to emergence of a theoretical framework from the data. It is important to mention that Strauss and Corbin (1990) do not completely dissuade people from reading literature in the substantive area before they start gathering data. However, they believe that some understanding of the

research area through literature study increases the theoretical sensitivity of the researcher when generating theory from the first data samples.

Glaser, however, argues that, only after the collection and coding of the first set of data, and after generating a preliminary theory in the substantive area, can one use data from a literature study of the substantive area to support the emerging theory (Glaser, 1992:31). In their book, Strauss and Corbin (1990:39) talk of preliminary theory being generated from initial data samples, which seems to agree with Glaser's idea. They ask researchers to use a theoretical framework that is generated from the initial data gathered as a starting point in their theory-building process. They stress, however, that the framework should not be derived from the literature study.

Strauss and Corbin (1990:55) also encourage the study of non-technical literature. They list non-technical literature as comprising of letters, reports, diaries, biographies, videotapes and various other materials. Non-technical literature can be used as sources of primary data, supplementing the data collected through interviews and observation. Glaser (1992), however, says that non-related literature can be used to sensitize the researcher theoretically and to improve writing style and presentation techniques, but strictly forbids any literature study in the substantive area of study, whether technical or non-technical. Possibly the debate is on their interpretations of what constitutes technical, non-technical, related and unrelated literature. Glaser (1992:35) summarizes the argument by noting that reading unrelated literature:

“...maximizes the avoidance of pre-empting, preconceived concepts which may easily detract from the full freedom to generate concepts that fit and are relevant when initially coding and analyzing the data as it is collected”.

The next section discusses how the present researcher used this debate on the use of literature in this study.

Use of Literature in this Research

Sociological researchers such as Glaser and Strauss (1965) often have the privilege of carrying out investigations in areas whose disciplines could be very new to them. An example of this is their study on the dying of patients in their paper, “Awareness of Dying” (Glaser & Strauss, 1965).

It is quite possible that they did not have prior knowledge of the literature in that particular discipline. However, this is not the case with many researchers, who investigate issues in their fields of expertise. As Campbell, (2009) noted, very few people, if any, begin their research without some prior knowledge. Prior knowledge and prior reading of the field under study is a given. He goes further to tell researchers to concentrate on reducing the impact of this prior knowledge on the process of theory generation. In such cases, it is difficult to heed the dictum not to read literature in the substantive area. This is the case in this research study. The author is a lecturer in Information Systems and is familiar with literature on software development and ontologies. It is apparent that the biggest challenge then is to observe and adhere to the requirements of the dictum laid down by Glaser and Strauss (1967).

The researcher instead allowed his prior knowledge to encroach into the research. This, if we use Glaser's argument on what constitutes data (Glaser, 2002), should not have an effect on the answers that the respondents gave during the interviews. The data should come from the field and the researcher limited as much as possible any discussion on software development and ontologies with the respondents before the interview. In cases where the interviewees wanted to know about the study, these discussions only took place after completion of the interview.

It was only after the analysis of the preliminary data samples that the researcher started consulting the literature in the fields of software development and ontologies. The researcher felt free to do this using the argument that GTM proposed by Glaser and Strauss is not a prescriptive method but should be viewed as a guiding principle.

3.3.3 Data Acquisition Methods

Interpretive research can employ both quantitative and qualitative data. The choice of the data to be gathered is guided in most cases by the philosophical underpinnings of the research problem. This section will briefly discuss the qualitative data gathering method only.

3.3.3.1 Qualitative Data Acquisition

The data gathering methods employed in this research are qualitative. There are four basic tenets of qualitative research: contextualism, process, flexible use of theories and, lastly, the acceptance and focus on the participants' and researcher's perspectives (Struwig & Stead, 2004). These aspects have to be considered whenever a researcher collects data. Qualitative data are usually unstructured, unbounded and textual in form. The data are a reflection of

how the respondent views his/her contextual settings and of how he/she interprets and presents the organizational view. The richness of these data may be lost when researchers attempt to aggregate or summarize them.

Qualitative research data collection can “capture complex and subtle social and behavioural data” (Cornford & Smithson, 1996:125). Using qualitative research, data are acquired by means of techniques such as interviews, questionnaires and observations. Characteristically, qualitative data are “narrative, impressionistic, opinionative and textual”(p.125). It is widely believed that qualitative research data collection methods are holistic in that they capture the most representative set of a study area.

3.3.3.2 Data Acquisition Methods for this Study

According to Olivier (2004:11), a research project should have primary and secondary goals. These can be deduced from the research objectives and the research questions for each project. Depending on the type of goal to be reached, either primary or secondary, different methods can be used to achieve it. For the sake of semantic uniformity these methods are classified as primary and secondary data-acquisition methods.

Prerequisites for GTM Data Gathering

“How you collect data affects which phenomena you will see, how, where, and when you will view them, and what sense you will make of them.”

Charmaz, 2006: 15

Data collection is a very delicate process that needs to be managed. The quality of the research results is directly dependent on the data-collection process used. Several data-collection techniques are available for qualitative research, but, for GTM purposes, it is important to give more weight to field notes, interviewing, document and report sampling. GTM requires all interviews and field notes that are collected at the beginning of the research to be transcribed for coding and analysis.

Charmaz motivates for a data-gathering method that would allow researchers to view the researched phenomena in the same way as the participants in the research area see it (Charmaz, 2006:13). The data-gathering techniques can be changed during the research process to suit events occurring in the field. In fact, the data-collection methods should be chosen so that one gets appropriate data that soundly answer the research questions. Later, the researcher may chose to transcribe code and analyze specific portions of the data, a

process called theoretical sampling. Theoretical sampling is done in an attempt to reach theoretical saturation and density in the generated theory (Glaser, 1992).

3.3.3.3 Primary Data Acquisition and Sampling

In this study, the primary goal was to find the empirical evidence that supports the claim that ontologies can capture the human elements inherent in organizations. The survey method of interviewing was used to gather data from system development practitioners. The only problem encountered is that the interviewees had no knowledge of IS ontologies.

However, a vast range of people could have been used as respondents: end users, system analysts, programmers and IT academics would all have met the criteria. However the present researcher decided to start the data-gathering process with three IT academics who also have varying levels of industry experience.

The reason for doing this was based on the need to fine-tune the interviewing process as well as the interview questions. Academics are naturally expected to have a wide general knowledge of software development, including the concepts and practices that are used in the development process. This is discussed in greater detail in Section 6.1.

The three IT academics have different backgrounds, including system development methodologies, the philosophy of information technology, soft systems development methodologies, systems thinking, traditional and agile methodologies, programming and system and business analysis. The choice of academics allowed the researcher to elicit as exhaustively as possible a good number of codes and categories of incidents during the open coding process.

After the initial interviews, the door was opened for other practitioners, amongst whom were project managers, systems and software developers, system and business analysts and system test analysts.

Interviewing Techniques

With or without a specific research question, Glaser (1992:25) advises researchers not to ask direct questions during interviews. This dictum is used in a bid to guard against the preconception of emergence of data. He also proposes three fundamental formal ways of approaching the interviewing process if researchers use GTM. The first requirement is for the researcher to probe people working in the substantive area in an attempt to find their concerns. As they work in the substantive area, these people will have different ways of

arriving at a solution. The researcher is then tasked with finding reasons for the differences in their approaches to a problem.

The second requirement is to find categories of incidents as they show themselves in the substantive area. As Glaser (1992) stipulated, the fundamental rule for GTM is that theory must be based on emergent relevance of categories.

For sound GTM practice, Gasson (2003) identifies two elements that support Glaser's proposition. In the first, he advises that patterns inherent in the observed empirical data should form the basis of theory. Unlike hypothesis testing, he adds that inferences and prejudices or the association of ideas should not be entertained when conducting grounded theory research. Suddaby (2006) supports this by advising researchers that, when GTM is employed, one cannot make truth statements about reality, instead, grounded theory is about insights and explanations about social relationships and how they can be used to construct reality. Hence, it is more appropriate to deal with propositions and not hypotheses.

In the third, he notes that the use of the Constant Comparison Method allows the emergence of theory. This theory, the codes and constructs (or categories and their properties) are constantly weighed against new data. Such constant comparison confirms that theoretical constraints are a by-product of – and are embedded in – the data. The primary data consisted of seven interview data samples which were collected using unstructured open interviews. The total number of interview questions was twenty six (26) and focused on different aspects of software development and implicitly on ontologies, as shown in Table 6.1.

3.3.3.4 Secondary Data Acquisition Methods

The researcher used a literature study in both software development and ontologies as a secondary source of data. This literature consisted of journal papers, conference papers, textbooks as well as email discussions. Although ontological research and the use of ontologies in software development are still in their infancy in South Africa, a number of research papers and proposed differential ontology models have already been published.

The secondary goal also consisted of building theoretical constructs that were finally used in the development of an ontology framework. The secondary goal supported the primary goal by providing a solid reference ground and the necessary theoretical sensitivity needed in generating the constructs from interview data. More so, it provided the necessary ontology information needed to fill up the gap in software development methodologies. As part of the

literature study, previous work done on system and software development and on ontology development was consulted. These are reflected in Chapters 4 and 5 respectively.

3.3.4 The Research Goals

Every research process should have research goals. These research goals can be classified as technical, social or philosophical in nature. Technical issues deal with the implementation of systems (Olivier, 2004). With respect to this research, technical goals might deal with the implementation of the ontology framework in software development. This is the task of a design science project as explained in Hevner *et al.* (2004). As this was not the purpose of the research, this was deferred for future research.

Social issues deal with the humanistic nature of any system or with its investigation (Olivier, 2004). Social goals require scrutiny of the social construction nature of software products and of how the ontology framework can capture these social aspects. These social issues form most of this research study. Most of the discussions on Chapters 4 and 5 addressed this research requirement.

Philosophical aspects of the study delve into the hidden assumptions made about the research under study. The fundamental questions are:

- What are we investigating?
- Is it objective or subjective?
- Is it a given or a product of the human or societal construction in which it is found?
- What constitutes the research?
- What actually are the problems bedeviling software product development and what do developers do about them?

These philosophical goals dealt with the basic paradigmatic tenets proposed here of moving from a mechanistic to a romantic software development approach. These tenets are used to justify the important role played by ontologies in software development. The mechanistic characteristics of software products are discussed in Chapters 4 and 6. These were elicited from technical literature as well as from the respondents. Chapter 5 provides information on the descriptions of ontologies, their social nature and some of the areas in which they can be applied.

The data-gathering methods indicated in Table 3.1 fall into one of the following categories: empirical, creative or tautological (Olivier, 2003). An empirical study was carried out in order to get primary data. This empirical study was based largely on interviews. The idea was to use this empirical method to generate and explore theory. The empirical evidence gathered was not used for testing theory but for the generation of theory.

Table 3.1 summarizes the data-gathering methods used in this research and the specific data-gathering goals of the project.

Table 3.1: Research Goals and Data Gathering Methods

Research Goals and Data Gathering Methods (<i>Adapted from Olivier, 2004</i>)				
Goal Type	Data Gathering Method	Goal Category		
		Technical	Social	Philosophical
Primary	Survey (Interviews)		Yes	
Secondary	Literature Study	Yes	Yes	Yes
	Model	Yes		
	Framework		Yes	Yes
	Arguments		Yes	Yes

Secondary data were used in this research to extend the theory, to fill the gaps in empirical evidence and also to increase the theoretical sensitivity of the researcher. The literature study provided much of the secondary data. This is a tautological type of evidence-gathering. Tautological methods are good at transforming inputs to reveal hidden characteristics that would otherwise not be obvious in the inputs prior to their transformation (Olivier, 2004). This is the purpose of Chapters 4 and 5 as reflected in Chapter 6.

In addition to the literature study, arguments and theoretical frameworks were used to develop new abstractions or theories. This creative type of method was employed to build the ontology framework and later, the ontology-driven software development approach that is described in Chapter 6 of this study. Although the process of model formation is included in Table 3.1, this aspect was not used in this research since it did not fall within the scope of this

research project. A model is more prescriptive and usually is a product of design science research project. The development of domain or process ontologies can be likened to this type of research goal.

3.3.5 Data Analysis in GTM

The generation of theory is a process of converting raw data into information. In this study, interview transcripts were coded, interpreted and subjected to several cycles of analysis to come up with the substantive theory. It is very important to note that many of the respondents were not familiar with the discipline of ontology as applied to software development. However, they were quite familiar with the information systems discipline that included the software development process. The issues they revealed as requirements for romantic systems development needed to be mapped onto the ontological concepts that formed the framework. This process was heavily based on a review of the literature in software development and in the field of ontology, as discussed in Chapters 4, 5 and 6.

3.3.5.1 GTM Coding

Grounded theory recognizes two types of codes: substantive and theoretical codes. The conceptual meanings that are given by generating categories and their properties comprise their substantive codes. These substantive codes are made up of the data patterns that are revealed in the substantive incidents during field data-gathering. On the other hand, theoretical codes comprise conceptual models of relationships that theoretically relate substantive codes to one another.

The ability to generate these codes is of the utmost importance during the generation of grounded theory. To achieve this, three basic types of coding are used: open coding, axial coding and selective coding (Glaser & Strauss, 1967; Glaser, 1992; Strauss & Corbin, 1990; Charmaz, 2006). These three types of coding are discussed below. It should be noted that a qualitative analysis piece of software called Atlas.Ti 5.2 was used in this study.

Open Coding

Open coding is a process tasked with the discovery of categories and their properties and which classifies them into themes or categories, at the same time looking for trends in the data (Gasson, 2003; Glaser, 1992; Glaser & Strauss, 1967). During this process, Gasson (2003) advises researchers to look for commonalities, associations and implied causality in

the elicited categories. The basic premise for open coding is that the research starts with no concepts at all. In the end, open coding has to establish core categories.

Open coding breaks the data down into incidents. These are further examined and analyzed to check for similarities or differences in the incidents thus generated. Glaser (1992:39) urges researchers to constantly check the “category or property of a category” indicated by the incident. The process of eliciting categories or their properties should be based on sound, unbiased judgments and on an unbiased view of the data.

Gasson (2003) and Strauss and Corbin (1990) argue that good open coding is informed by literature.

The Constant Comparative Method

This is an analysis process in which the researcher constantly compares incident with incident and then incident with concept. The properties of categories are generated during this process. The next step is to find the categories or the properties of the categories to which the incident belongs. Suddaby (2006:636) noted that the constant comparative method is tasked with translating the raw observations to a higher level, to “more abstract theoretical categories.” This necessitates a continuous interplay between data analysis and collection. As Suddaby (2006:636) affirms, the most important and challenging part of employing GTM is the failure of researchers to “lift data to a conceptual level” resulting in mere reportages of data.

When generating categories, the grounded theory (GT) analyst looks for patterns and a conceptual name (i.e. a category) is given to a pattern of many similar incidents. The compared incidents can be used as indicators of the same concept. Saturation point is reached when many interchangeable incidents are found and coded. Categories are generated from similar occurrences of a pattern and not from isolated single occurrences that cannot be generalised. Then, theories should be allowed to emerge freely from the data. Patterns of data should be allowed to show themselves at a grand level of incidents and sift down through to the properties of incidents.

Lastly, when doing open coding, one can consider line-by-line analyses of sentences, paragraphs or entire documents. The most important thing is to allow the piece of data under analysis to form a comprehensible pattern or incident that can be used at the conceptualization stage to generate, discover or develop a category.

When researchers use the constant comparative method during GTM open coding, there are four basic questions, as expounded by Glaser and Strauss (1967), Glaser (1992) and Charmaz (2006), which should be used as lenses to direct the elicitation of categories as well as their properties. These questions are:

- Of what are these data a study?
- What category or what property of category does this incident indicate?
- What is actually happening in the data? And lastly,
- What are the basic social psychological processes or social structural processes in the main problem that make life viable in the action scene?

The questions should be asked repeatedly until such time as enough data, coding and analysis has occurred for the researcher to embark on “theoretical sampling and selective coding” (Glaser, 1992:51).

Unit of Analysis

Coding is a process of analyzing data. As such, the researcher is faced with the problem of choosing the correct unit of analysis. In open coding, this may be “*a sentence, a line from a transcript, a physical action ... or a combination of*” such elements. In data analysis it is important to differentiate between the terms used by the respondents and the technical terms that the researcher associates with the phenomena. Gasson (2003:82) claims that this will reduce the bias that could be introduced in the analysis by the researcher’s own preconceptions.

Axial Coding

Gasson (2003) describes axial coding as a constant search for relationships that exist among coded elements. Categories, sub-categories and their properties, as elicited during open coding should be scrutinized to check for similarities and dissimilarities in their associations. It is an attempt to relate structure to the process.

Theoretical Sensitivity

Glaser (1992) refers to theoretical sensitivity as the researcher’s knowledge, comprehension and expertise that enables the generation of categories and their properties. Strauss and Corbin (1990:31) regard this theoretical sensitivity as more of a personal trait. These qualities

enhance the researcher's ability to relate the categories and properties to hypotheses and later to integrate them into hypotheses according to the emergent theoretical codes (Glaser, 1992). The main task of theoretical sensitivity is to generate concepts from the data and to establish their relationships using normal models of theory. It is a case of finding meaning, relationships and concepts in the data collected.

The ability to undertake theoretical sensitivity marks the difference between an informed and knowledgeable researcher and a theoretically competent grounded researcher. Unlike the former, the latter has the ability to generate hypotheses and convert them into theory. Strauss and Corbin (1990) attribute this ability to the researcher's intelligence, research, academic and professional background, as well as to the researcher's understanding of the area under study.

Besides undertaking sociology classes on theoretical coding and conceptualization, Glaser urges the grounded theory researcher to constantly study "substantive and formal theory" in any discipline other than that under study. He urges researchers to "Study theory constantly", (Glaser, 1992:28). By knitting the theoretical codes together the researcher will be able to see the research, the research data and the concepts that emerge from the data in a novel way. These can then be used for the generation of theory.

Researchers should always bear in mind that GTM is not a verification type of methodology. Hence, the hypothesis so generated "need not be verified, validated, or more reliable" (Glaser, 1992: 31-32). He, however, stresses the importance of the "rigor of systematic generation of theory".

Theoretical Memos

While doing coding, the researcher is intermittently struck by emergent theories, theoretical formulation and ideas about data. These revelations should be documented and are referred to as theoretical memos. By documenting these emerging materials, the researcher can obtain an insight into the type of questions and data that still require exploration.

Selective Coding

The purpose of selective coding is to factor in data that implicitly and explicitly support the categories already identified and their properties. The researcher should step back and look at the research questions to find what the research data need to generate. At this juncture, the

researcher chooses data that support the intended theory and should realize that not all data are worth analyzing (Gasson, 2003; Glaser, 1992).

3.4 The Double Mapping Principle

The double-mapping principle is a concept that is put into practice when researchers are faced with a study in which two disciplines are both considered as substantive areas in a GTM study. As Goede (2005) found out, in areas where a research straddles two fields, for example, in data-warehousing and systems-thinking, the chances are that the respondents may not be familiar with one or other of these fields. Two options are thus possible: to collect data separately from people in both fields and match them during analysis or to interview a sample of people in one of these fields, depending on the substantive area of study and then do a theoretical mapping.

It is very important to note that, in this study, many of the respondents were not familiar with the discipline of ontology as applied to software development. However, they were quite familiar with the software development discipline. Although both fields of software development and ontologies are considered as substantive areas in this particular context, the researcher could not get responses that were ontology-related.

The respondents gave descriptions of software development issues using their domain language. To match the categories of incidents to the ontology concepts, a double transformation (mapping) had to be done, as illustrated in Figure 3.1. However this mapping required the findings from the ontology literature study, as reflected and documented in Chapter 5, to be used as a primary document. As a primary document in Atlas.Ti, the findings in Chapter 5 generated ontological categories that were later mapped to the software development categories as generated from analyses of the interview data and literature study in Chapter 4.

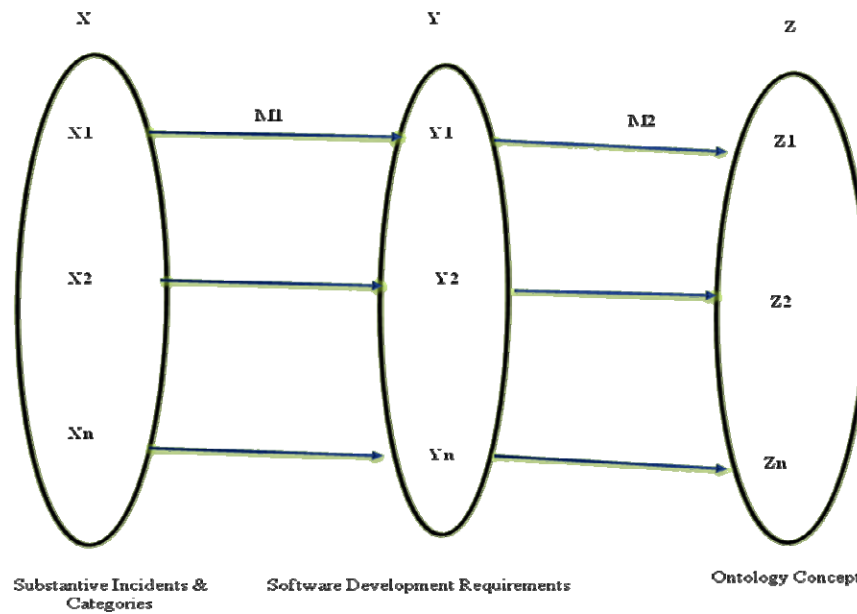


Figure 3.1: Three-Tier Incident and Concept Mappings

In Figure 3.1, X is a class of incidents and categories of incidents in the software development field. Y is a class of codes, requirements for software development process. M1 is the first mapping $M(X)$ to $M(Y)$. In a study consisting of a single substantive area, the relationship between categories in $M(Y)$ could singly be used to generate theory.

However, in this study, a second mapping, M2, was required to map the software development categories to the ontological concepts to which they relate, that is, the transformation of $M(Y)$ to $M(Z)$. This second mapping became necessary because of the existence of the second substantive area and the research requirements of finding a framework of ontology aspects that could be used to improve the software development process. These ontology concepts do not directly have the same meanings as the concepts in use in current software development practices.

The implication of this double mapping is that the relationships between categories generated through mapping M1 could be distorted by the second mapping M2. This could have a serious implication on the final theory generated, as reflected at $M(Z)$. To have a sound data analysis in a case like the one described here, researchers are urged to have a thorough knowledge of literature in the two areas under study. The knowledge of literature, in addition to increasing theoretical sensitivity, will also ensure a more appropriate mapping of concepts and their categories through mapping M1 and M2. In short, researchers are urged to conduct a thorough and continuous study of literature.

3.5 From Substantive to Formal Theory

It is important to note that the theory that is generated during any well planned grounded theory research should support the relationship of the data under study in the substantive area. It is claimed (Glaser & Strauss, 1967; Glaser, 1992; Strauss & Corbin, 1990; Gasson, 2003) that, if the theory fits the data, it is a more appropriate representation of the patterns inherent in data for that same area. This theory is defined as substantive theory. Strauss and Corbin (1990:173) describe substantive theory as one that is generated from “*the study of a phenomenon situated in one particular context*”. This theory, which is generated from a single research project, i.e., substantive theory, cannot be generalized and expected to apply snugly to many other areas with different contexts.

With time, substantive theory needs to be improved to form formal theory. Formal theory is the product of a cross-sectional study of many different contexts. This can only be done by asking researchers in the field to continuously conduct research in the same area and to add more knowledge to improve the representativeness of the theory in the substantive area, but in different scenarios. This process takes time and sometimes requires different types of researches and different researchers. When this is done, the theory can now be generalized at a more conceptual and abstract level. When done soundly, the theory can be referred to as formal theory (Gasson, 2003).

Generation of this formal theory requires time, with different researchers making use of the same substantive theory and application of many conceptual reflections and abstractions before it can be accepted as such in the specific discipline. Also, some people use the variety of situations studied to distinguish between substantive and formal theory. The reflexive type of process that was followed during this study is discussed below.

3.6 The Reflexive Grounded Theory Process

The grounded theory process does not follow a linear set of steps in practice. It is a repetitive process that requires the researcher to consider the previous process as well as the next process. Depending on the results achieved, the steps followed can always be adjusted to improve on the process. This is termed the reflexive process.

The researcher, therefore, followed the reflexive approach discussed by Gasson (2003). The reflexive approach, as used in this research project, is shown in Figure 3.2 below. From project initiation down to the generation of substantive theory, the researcher needed to reflect and make explicit the influences of assumptions and actions taken at each stage of the

research process (Gasson, 2003). The research process is portrayed in a non-linear fashion, a characteristic of how the researcher accomplished the research study.

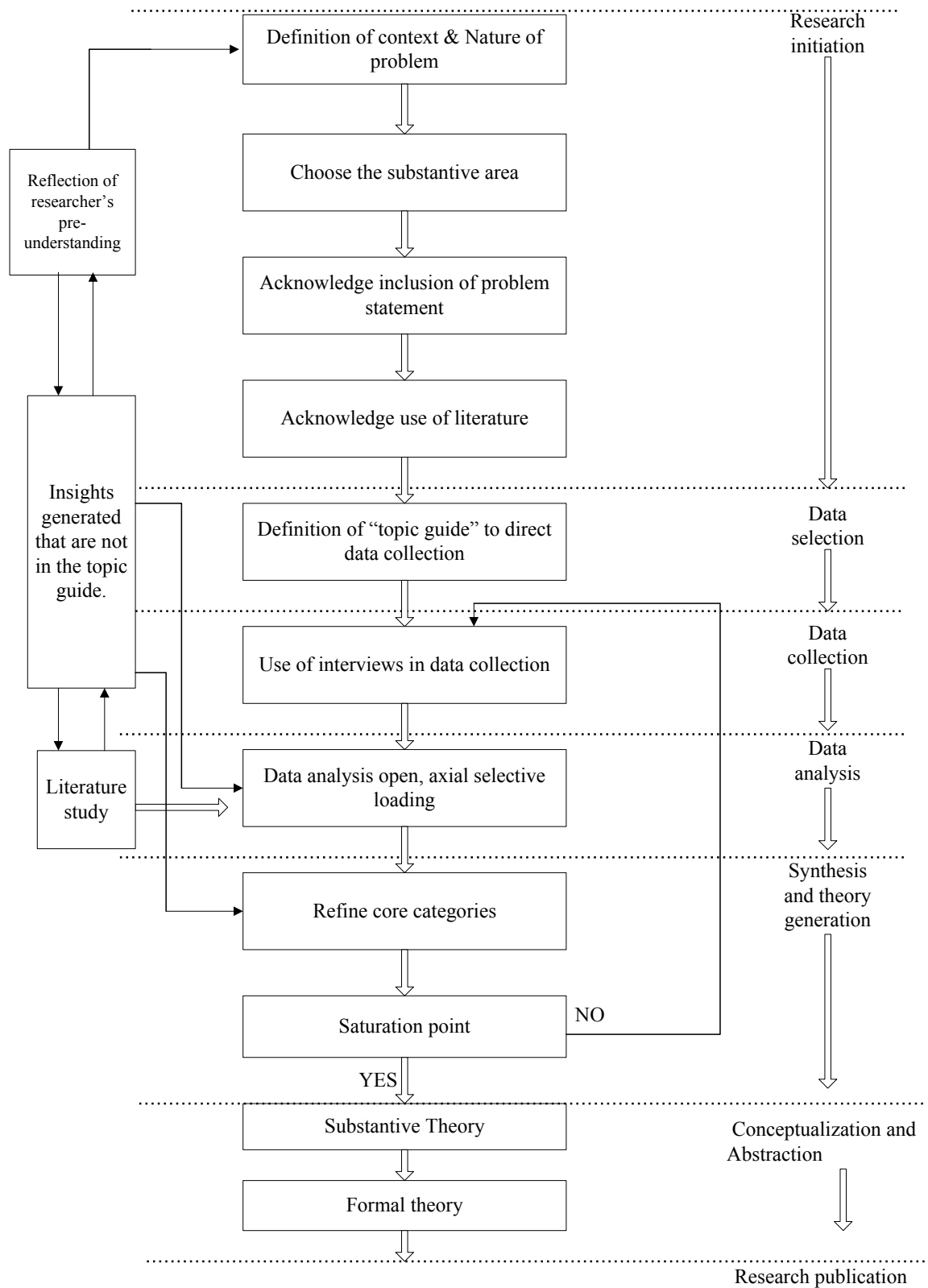


Figure 3.2: A Reflective Grounded Theory Process (Adapted from Gasson, 2003)

In brief, the researcher started by defining the problem context and its philosophical nature. The substantive area of research was identified at the same time. Propositions were included together with highly generalized research questions. In addition, the use of literature and of the researcher's pre-understandings was acknowledged.

The next phase included selection, collection and analysis of the data. These were not done in a linear fashion but in a cyclic way. At each phase of the research process, the researcher had to reflect on the steps used, compare these with the goals of the research and evaluate the impact of that phase on the rest of the study. The influence of individual phases on previous and subsequent phases also had to be analyzed. At the end the substantive theory was developed as "An Ontology-Driven Software Development Framework" that guided the development of the ontology-driven development approach and methodology in this study.

3.7 Judging the Fidelity of Generated Grounded Theory

Since quantitative and qualitative research strands are sometimes dichotomous, it would be unwise for researchers to use quantitative criteria for assessing the rigour and quality of qualitative research. Firstly, the assumptions made about the two types of research are very different and hence the perceptions of reality in these two paradigms also differ. When GTM is used in a qualitative research, the positivist notions of "falsification and hypothesis testing" are not accepted (Suddaby, 2006:634). Glaser (1992) adds that the hypothesis generated does not require further testing or validation. It should be borne in mind that this method is grounded in the idea that social theory must emerge organically, depending on the good fit of the observed data to the conceptual categories identified by the researcher during the coding process. This is supported by Hevner *et al.* (2004:87-8) who noted that research rigour is judged by the "adherence to appropriate data collection and analysis techniques". This is heavily encompassed in the systematic data-gathering process and in the analysis of these data using the constant comparative method. In short, the data should be able to explain and predict successive interpretations and, lastly, the relevance of categories to the core issues under observation. It is important to note that GTM is more interested on how "subjective experiences can be abstracted into theoretical statements about causal relations between actors" (Suddaby, 2006:635). It is, therefore, important to find criteria for judging qualitative research that fits the interpretive nature of such problems while, at the same time, keeping the appropriate measurement characteristics that allow rigour and quality to be judged in the same way as in the quantitative studies.

There are four dichotomous issues that usually confront researchers in their attempts to judge and qualify the fidelity of the research process. These issues, taken dichotomously as quantitative versus qualitative, are: objectivity *versus* confirmability, reliability *versus* dependability or auditability, internal validity *versus* internal consistency and, lastly, external validity *versus* transferability (Gasson, 2003). During a research study, these issues are closely related to the research design process and ultimately affect the quality of the research process, research results and their acceptance in a discipline as new contributions to a body of knowledge.

This section discusses the aspects that apply to qualitative interpretive research only and the way in which they were incorporated with respect to this research. Together with the factors discussed by Klein and Myers (1999) (*see Section 3.9*), these issues will be revisited in the concluding chapter of this research.

As shown in Table 3.2, four criteria are proposed by Gasson (2003) for use in checking and maintaining quality and rigour in a qualitative research study. These criteria: confirmability, dependability, internal consistency and transferability, together with their core issues are described below.

Table 3.2: Criteria for Judging Qualitative-Interpretive Research

Criteria for Judging Qualitative-Interpretive Research (<i>Adapted from Gasson, 2003</i>)	
Criterion	Core Issue
Confirmability	Findings should represent, as far as is (humanly) possible, the situation being researched rather than the beliefs, pet theories or biases of the researcher.
Dependability/Auditability	The way in which a study is conducted should be consistent across time, researchers, and analysis techniques.
Internal Consistency	How we ensure rigour in the research process and how we communicate to others that we have done so.
Transferability	How far researchers may make claims for a general application of their theory.

3.7.1 Confirmability

This aspect requires the research findings to depend on the subjects and conditions of the study rather than on the researcher (Gasson, 2003). Confirmability can be achieved by having a thorough and complete description of the research and data-collection processes, analysis

and of the findings, according to the research context. In the research report, documentation and explanations of how the results were obtained and of what influenced specific decisions to pick on certain data patterns should be included.

In this study, in addition to the subjects, the researcher is also included as part of the research. Therefore, a collective assessment of both the subjects and the researcher's interpretations is needed.

3.7.2 Dependability/Auditability

This factor requires researchers to constantly visit and question the assumptions made about the research problem, from the start of the research through to the end of the research process. They need to document and justify all the decisions made about the research process. This process should enable an outsider, trained to think like the researcher at the time of the research (i.e. having the same context), to be able to arrive at the same conclusions using the same data from the field. In short, researchers should provide an audit trail to anybody interested in the research to check the steps followed in the process.

In this research, the accepted ways of using GTM in qualitative research were followed. In cases where adaptations were made to the requirements for sound grounded theory research, the assumptions and reasons for making such changes were documented.

3.7.3 Internal Consistency

This issue deals with the credibility of any qualitative research. It is important to have a reflexive process of data coding and analysis as depicted in Figure 3.2. A critical analysis of each stage was done and justified. The appropriateness of the data used to generate the substantive theory should be discussed relative to the theoretical findings (Gasson, 2003:95). The use of the constant comparative method in this research study is discussed in order to explain how the selective coding and theoretical sampling processes gave rise to the research findings. By the use of this method, the biases and distortions in the data analysis could, at least, be identified and minimized.

3.7.4 Transferability

This notion calls upon researchers to look for '*transferability and fit*' between those contexts in which the generated theory or results may be applicable. In this research, after generation of the substantive theory, a further step was taken to formalize the theory through a process

of abstract conceptualization such as that proposed by Strauss and Corbin (1990). In support of Gasson (2003)'s quality criteria as discussed here, the following section discusses other criteria for improving the quality and fidelity of qualitative research (Klein & Myers, 1999).

3.8 Evaluation of Qualitative Interpretive Research Study

As discussed in Chapter 2, the underlying epistemology of this research revealed it as an interpretive study. Working from this background, it is highly important to discuss Klein and Myers' seven principles (Klein & Myers, 1999), which they regarded as important in the evaluation of an interpretive research study. When discussing these principles, the present author accepts them as guiding principles only. He agrees with Klein and Myers that they should not be used and applied mechanistically. In any case, interpretive research can never have a "pre-determined set of criteria" that can be used in the research. The execution of the research is guided solely by the context of the study.

In addition to discussing these seven principles, the researcher also discusses the relevance and application of each principle to this study. Reference is also made to sections in the thesis where each principle is applied.

3.8.1 The Fundamental Principle of the Hermeneutic Cycle

This principle supports the idea that human understanding can only be achieved through repeated cycles of interpretation. Starting with one's fore-knowledge and prejudices, understanding of any situation involves an interdependent process of interpreting individual cases of the situation and knitting them together to provide a holistic understanding of the whole.

In this research, Chapters 2 to 6 were developed and reported using this principle. The research philosophy was discussed in Chapter 2. The research design is discussed in this chapter. Chapter 4 covers the investigation of literature in the field of software development. Different cycles of interpretation were done to come up with issues that characterize current mechanistic software products.

Chapter 5 contains a synthesis of literature on ontologies and their relevance to software development. In Chapter 6, based on the philosophical underpinnings of grounded theory, the researcher discusses the empirical data gathered from the respondents. This involved defining categories and their properties, employing the process of theoretical sensitivity and

axial coding to abstract relationships among the categories and their properties and, finally, using the constant comparative method, generating the inductive grounded theory.

This principle basically is fundamental, not only to the other six principles as discussed by Klein and Myers (1999), but also to the whole research study discussed in this thesis.

3.8.2 The Principle of Contextualization

Klein and Myers (1999) noted that the situatedness of a research study, based upon the respondents' and researcher's interpretation of the scenario, has an effect on the findings of the research. To properly understand the study, the context upon which the research is based should be discussed and elaborated. Context is built by reflecting on the social settings of the people involved in the research, as well as by taking their historical backgrounds into consideration.

This principle was applied in Chapters 2, 3 and 5. Of importance is the discussion of this chapter, which spells out the unique environmental, social and practical conditions that were considered in this study. Section 6.1 on research profiles can also be used as evidence of this principle.

The conditions are referred to as unique because they influence only the present research study and the results generated are highly dependent on this context. It is also important to note that the discussion of the research results and the conclusions based on the findings are grounded heavily on this research design and the researcher's theoretical sensitivity. Another researcher whose theoretical sensitivity is different from this researcher may reach a different conclusion.

3.8.3 The Principle of Interaction between the Researcher and the Subjects

Interpretive data are a product of the interaction between the researcher and the researched. The researched, in addition to software development phenomena, also includes the respondents interviewed during this study. It is important that the relationship between the researcher and researched be critically reflected upon and annotated.

In this regard, we reflect on the choice of the respondents, the data-gathering techniques used (in this case interviews) and on the conditions that existed during the interview process. In some cases, it is also prudent to reflect on individual interviews and on the reasons, if any, for tuning each interview relative to the type of respondent.

Klein and Myers (1999:73) urge researchers not to ignore the role played by respondents as “interpreters and analysts” of data. In this research, we acknowledge our choice of respondents, as discussed in Section 3.3.3.3, and also the fact that many of the respondents needed clarification of many of the concepts that were used by the researcher during the interviews.

As the respondents appropriated the new meanings of these concepts, they had to adjust their understanding of both the software development field and the research problem. As a result, they also assumed the position of analysts in this study.

3.8.4 The Principle of Abstraction and Generalisation

This principle makes use of the first two principles, the principle of the hermeneutic cycle and the principle of contextualisation. It calls upon researchers to relate the field study results to the theoretical concepts that describe the phenomena under study.

The philosophical grounding in GTM, as discussed in Section 3.4, also supports the application of this principle. In generating categories and the properties of these categories, the concepts used need to be mapped to ontological theory concepts so as to constitute the ontology framework for software development. This principle is used in Chapter 6 especially during the open, axial and the application of the double mapping principle. In addition, being theoretically sensitive, as discussed by Glaser and Strauss (1967), Section 3.4.5.1 calls upon the researcher’s skills in abstraction and generalisation.

3.8.5 The Principle of Dialogical Reasoning

Interpretive research using GTM requires the research to meet specific methodological requirements as discussed in Section 3.4.1. These requirements relate to the use of literature, theoretical sampling and research questions.

In principle, the researcher is urged not to begin with a problem statement but only to have an interest in the chosen field of study. This obviates the need for research questions and minimizes the possibility of the introduction of researcher bias. The researcher is also urged not to read any of the literature in the substantive area of study.

During data collection, the researcher is directed to choose those respondents who are most likely to provide answers relevant to what is being researched. The interview questions can also be changed and adapted as the study progresses, focusing on the issues that need to be addressed, for example, by filling the gaps in the generated theory until saturation point is

reached. This process is termed theoretical sampling and works together with selective coding in a grounded theory research. This principle is addressed theoretically in this chapter and empirically in Chapter 6.

3.8.6 The Principle of Multiple Interpretations

This principle urges researchers to accept differences in how their respondents interpret the same phenomena. Based on their background knowledge, different people give different meanings to things, depending on their situatedness and contextual setting. These different interpretations should be annotated and described. The reasons for these discrepancies in interpretations have to be inferred from the context of the respondents.

These differences should not be construed as conflicts but as differences in viewpoints. They should be used by the researcher as instruments to aid his or her further understanding of the researched phenomena. The selection and use of interviewees from different software development specializations also ensured that different perspectives are gathered according to the different software development phases in which they are working. There are several of these multiple interpretations in this research, which were used to increase the researcher's theoretical sensitivity.

3.8.7 The Principle of Suspicion

In all data-gathering exercises random and systematic errors are evident. If undiscovered, these will affect the quality of the research findings. The principle of suspicion requires researchers to look for “possible biases and systematic distortions in the narratives collected from the participants” (Klein & Myers, 1999:72). It should be noted that these biases can also come from the researchers themselves during the interview and analysis stages. The principle of constant comparative analysis ensures that biases and distortions are reduced. Also, the researcher replayed all of the interview audio recordings during data analysis in order to increase his understanding of the data as well as checking the fidelity of the transcriptions.

3.9 Limitations of the Research Methodology

This section discusses some issues that may affect the findings from this research. The qualities of the various elements of the research are also discussed from the perspectives of Gasson (2003) and Klein and Myers (1999). Firstly, the quality of the data-gathering process was affected by the fact that the researcher considered too wide a spectrum of software practitioners. The researcher should have sampled at most one or two categories of software

practitioner types, such as developers, programmers, project managers, systems analysts and users. This, however, may be regarded as an advantage, since the varied types of practitioners brought a broader perspective to issues affecting software development.

Secondly, the research might have been improved if many of the respondents had prior knowledge of ontology as used in systems development. This lack of ontology knowledge added a second layer of analysis to the process in the researcher's attempt to match software development concepts to ontology concepts. In such a process, some of the rich information may have been lost. The plus side of this, however, is that lack of prior knowledge in ontology helped to reduce the bias and preconceptions of the respondents.

In terms of judging the quality of this research, the researcher discovered that some of the criteria proposed by Gasson (2003) and Klein and Myers (1999) could be used in conjunction with each other.

3.10 Evaluation Criteria for the Generated Theory

There are several evaluation criteria that are used worldwide in different institutions and various academic disciplines (Introna, 1992; University of Victoria, n.d.). However, it would be futile to try and include all such criteria in the evaluation of a single thesis. On that note, the following criteria were chosen as being more representative and form a very subjective selection:

- a. Thesis integration and coherence – A thesis should be logical, and have a golden thread allowing continuous rational connections between the different subsections, sections and chapters of the document (University of Victoria, n.d.). It should not be a concatenated grouping of unrelated sections and chapters.
- b. There should be sufficient study of relevant literature – A research study that culminates in a thesis should be grounded in a sound study of literature, emphasising recent developments in the substantive areas of study. Thorough knowledge of original literature sources, the fields under study and both the theoretical and methodological groundings to the study should be achieved and reflected in the thesis. While the research should not be penalised for not covering exhaustively all literature in the field, it should be borne in mind that “a critical, analytic approach” to literature study must be followed. In such a study, different academic discourses should be acknowledged and that possible contradictions in views, both theoretical and

methodologically should be noted. It should also take into consideration any possible sources of error in the interpretation and use of the literature in the study.

- c. Statement of the research problem – Depending on the research approach adopted, the research problem statement may be developed from the study of literature. However, using GTM, the research interest, instead is considered. The problem can only be noted after the analysis of that first batch of data. The problem statement can then follow under any of the three categories, namely:
 - Proposing a novel theoretical or methodological slant on a topic area;
 - Development of a new area where a theory(method) can be applied; or
 - The problem statement can be presented as research questions or propositions.
- d. Suitability of the methods of enquiry adopted – This covers the theoretical underpinnings and methodological underpinnings of the research. When considering theoretical groundings, the researcher is urged to look at the claims made about reality, the perception given to what is being researched. The ontological, epistemological and humanist stances taken about the nature of reality should be brought to the fore, clearly and succinctly. These underpinnings can be considered as a lens under which the research problem is observed. On the methodological grounding, the researcher should clearly document all the steps, giving reasons for data-gathering, analysis and presentation of the results. It is common to allow a slight diversion to discuss other research methods that could be possible candidates of the research enquiry. This allows the research to build enough ground to support the chosen research approach. Claims to a better methodological approach should be corroborated by explanations of why other methods were discarded. It is also important to have sound linkage of the methodological grounding to the theoretical grounding.
- e. Analysis of data – The methods of analysis should be aligned and suitable to the research method chosen. In addition to documenting all the analysis steps chosen, in the analysis process particular attention should be paid to the assumptions made and to new revelations arising from the process. Lastly, only important summary data should be included in the main body of the research report, with primary and secondary data being relegated to the appendices (University of Victoria, n.d.)
- f. Theory should unify various, previously unrelated problems or concepts (Introna, 1992).

- g. The theory should produce a new perspective on existing problems. This should lead to new understanding of the persisting problems under investigation. (Introna, 1992).
- h. The theory should produce unconventional ideas. These new ideas could be radical and challenge existing beliefs in the field of study (Introna, 1992). Stoke and Bird (2008) encourage researchers to use the '*minimal creativity*' as a yard stick. At the minimum, two conditions should be checked: novelty and agency. In their words, an idea, **F**, may be considered novel if, and only if, **F** has never occurred before and it is new to the history of ideas in the study discipline. In this case, we assume a historical novelty that is judged using "socio-technical facts, not behavioural ones" (Stokes and Bird, 2008:230). With agency, it can be stated that, "some behaviour, artefact or event **F** is the product of the agency **A** only if **F** would not have occurred had **A** not acted in some autonomous way" (Stokes and Bird, 2008:231). Creativity, therefore, at the very least requires "the right kind of agency and novelty" to come together in a research study (Stokes and Bird, 2008:232).
- i. The theory must exhibit positive and negative heuristic power. Heuristic power relates to the ability of the theory to guide users in formulating new research problems as well as solving some other problems (Introna, 1992).
- j. Contribution of theory to body of knowledge – Publishability of the theory is often used as a criterion for judging contribution to the body of knowledge. However, it is also important to check, compare and contrast the assertions that are made by the generated theory with those of other previous researchers in the same field (Introna, 1992). At least, the theory can improve on the precision of these assertions and the explanations thereof.

These evaluation criteria will be used in Chapter 7 together with the criteria used by Klein and Myers (1999) and Gasson (2003), as discussed above, in order to judge the contribution of this research study to knowledge, the practice of software development and to the information systems discipline in particular.

3.11 Summary

When going on a journey, travellers should always ensure that they have chosen the correct mode of transport and have the financial support and the appropriate accessories for use during their journey. Most importantly, they need to identify their destination correctly. This chapter acts a route planner for our research journey and includes a set of methods, techniques and tools needed for successful completion of this research journey.

The chapter started with a revisit to Chapter 2, in which we elicited the philosophical nature of the research problem. Then, the methodological requirements of the study, that is, the research method used, the design, the data-gathering and analysis techniques and tools were identified. The chapter concluded with a discussion on the quality considerations of the study. It is also noteworthy to discuss AtlasTi.5.2, the text analysis tool used in this project. However we deferred the complete discussion of the Atlas.Ti 5.2 modules to Chapter 6, which is dedicated to data analysis. The discussion has been very silent about how the research will finally reach theoretical saturation. According to Suddaby (2006), researchers should not focus on telling the reader about saturation point. He notes that theoretical saturation is not always obvious in most GTM researches. There are other basic tenets to GTM that should be followed and described in a research thesis, such as adherence to theoretical sampling, and constant comparison (Suddaby, 2006) that provides a cue to the quality of the theory generated. However, as noted in Brown (2008), the use of analytical software tools such as Atlas.Ti can hinder or distort such reportages. Most of this documentation is captured in the software repository leaving the author to document the results that lead to inductively generated theory. The next chapter is dedicated to discussion of software development practices.

CHAPTER FOUR

SOFTWARE DEVELOPMENT PRACTICES, INFORMATION SYSTEMS AND ORGANIZATIONS

4.0 Introduction

Chapter 1 gave a roadmap for this thesis. Chapter 2 discussed the philosophical groundings that guided the choice of the research methodology that was later discussed in Chapter 3. In this chapter, the concepts of software development, software development process, software development methodology, mechanistic products and romantic products that were briefly discussed in Chapter 1 are further explained in the context of their use in software development projects. In addition, this chapter also looks at the theoretical groundings, that is, the activity system, actor network system and the theory of organized activity that inform us of the world view that we should subject our organizations and their organizational information systems to.

A software development process is required to produce a piece of software. The software development process can be viewed as a framework or structure that is used during the development of a software product. Some examples of the software development processes are the waterfall, V-model, iterative and spiral models as discussed in Schach (2005) and Pressman (2005:79-88). The software development process prescribes the activities and techniques at times together with the tools that must be used to accomplish the different software development activities. There are varied views to what is software development. For example, Kirlidog and Aytol (2010) and Brooks (1995) view software development as a piece of creative art. This implies that some traits cannot be automated as they heavily rely on the developers' personal characteristics. On another note, Hirschheim *et al.* (1995) regard information systems development (ISD), as the use of information technologies in solving organizational problems. Software development is therefore part of ISD since software products are needed to implement information systems. This process is very complex and "cannot be captured in some formal model" only but requires the inclusion of some social approaches in its solutions (Hirschheim *et al.*, 1995:2).

In terms of players, the software development field is characterized by two major groups of stakeholders: the software developers and the method engineers (Gonzalez-Perez & Henderson-Sellers, 2006). Software developers may be any one of the following: system

analysts, programmers, business analysts or system architects. On the other hand, method engineers define and prescribe the methodology to be used by developers in their quest to construct software products. It is important to note that this researcher is assuming the role of a method engineer by attempting to develop a new software development approach. These two terms, that is, software developers and method engineers, refer to the roles played by the individuals or organizations involved in the software development process.

On the other side of software development, there are also information systems. These consist of three parts: the formal, informal and the technical parts. In current development approaches, the software product is usually part of the formal part of an information system. It is the software products, together with the hardware part of the technical subsystem, that are usually tasked with the day-to-day running of information systems. The formal subsystem has a bureaucratic nature. In it, form and rule replace meaning and intention.

Knowledge is a key factor in the development of software products. This knowledge is found in the organizational structures of the system to be developed. Any information system development methodology, therefore, should be able to transfer this knowledge and share it with the actors in the organization. In addition, it is only in a practice that this organizational knowledge is found. Strictly speaking, the knowledge is embedded in the situations where people perform a practice. For this reason, since information systems should capture and share this knowledge, the tacit nature of the knowledge makes it difficult for system developers to explicitly understand the task of software product development. This chapter will discuss some issues that limit the capturing and transfer of knowledge in organizational information systems.

Descriptively, the informal subsystem consists of a sub-culture in which meanings are established, intentions understood, beliefs are created and commitments and responsibilities are made, altered and discharged. This constitutes the tacit and implicit rules, procedures and power structures that are inherently a permanent attribute of the human factor. The informal subsystem has never attracted much attention from software developers.

In the end, since the informal subsystem is very fuzzily defined, the technical subsystem is used to automate the formal subsystem only, which is only a part of the whole system. This chapter starts with a brief excursion into the nature of organizations and information systems. It briefly addresses the influence of culture, concepts and context in an organization's

practice. The practice of software engineering is then discussed, focusing mainly on the dictates of the definition of engineering in the field of software development. Together with the theory of organized complexity (TOC), systematicity and system formation, this is used to explain why current software development methods result in mechanistic products.

4.1 The Nature of Organizational Systems

There is currently a major difference between the nature and representation of organizations. In an attempt to match and reduce the gap between these two, three theories, the theory of activity systems (AT), actor-network theory (ANT) and the theory of organized action (TOA), are used to assess organizational systems. This assessment is used to support the argument that organizations and information systems have a socio-technical nature. These theories all together will combine to motivate for romantic world view which is the theoretical grounding for this study. As discussed earlier, sound theoretical and methodological foundations are at the core of a sound research investigation. A theoretical framework should be conceived as a structure that holds together and supports theory of the study. It must be able to explain why the problem exists and must justify the need for research. It is from these theoretical groundings that a summary framework of what makes up a development approach will be deduced.

People, information systems and the environment in which organizations exist can be considered holistically as a social group of actors interacting through networks. As Ngwenyama and Lee (1997) contend, being social involves the alignment of an individual's actions to both the organizational context and the other actors involved in performing a social action. All social interaction is governed by a social culture. This culture has to be observed and studied during system development.

4.1.1 The Complex Nature of Organizational Systems

Organizational systems are examples of dynamic and complex systems. The complexity of these systems can be measured using the concept of requisite variety (Rosenkranz & Holten, 2007:57). Requisite variety views organizational systems as possessing several possible states, in terms of "patterns of behaviour" or "number of manifestations". During software development, it is the developer's intention to capture and maintain these patterns of behaviour (manifestations) in the resultant software product. However, during the software development process, the tasks of modelling, that is, of developing the analysis, design and

implementation models tend to reduce the complexity of these organizational systems by reducing their requisite variety. This, in turn, reduces the possible behavioural states of the subsequent software products and information systems under development. This process is regarded as the reductionist principle. Reduction in the possible behavioural states and, hence, in the requisite variety of the original system, in turn reduces the life responsiveness of the modelled and developed system. It is this researcher's opinion that most information systems fail to provide value to their organisations because of this reduction in requisite variety.

In order to maintain the requisite variety of organisational systems and to transfer them to the developed systems, two fundamental things have to be done. Firstly, either the modelled system has to have its requisite variety reconstituted to the original un-modelled state of the organizational system and, secondly, the original system should never be modelled using the reductionist principle. An alternative would be to allow the implementing tools of the system, as well as the system users, to possess as much requisite variety as that which existed in the original un-modelled system (Rosenkranz & Holten, 2007). Practically, however, it is impossible to have tools and users that have the same behavioural modes as the original system. In the end, the only practical way is to find methods and tools that will maintain the requisite variety during the process of system development. However, Lemmens (2007:57) notes that the failure to develop tools that maintain system behavioural characteristics has curtailed the development of systems that can capture human aspects. This is one of the several research goals of this study.

4.1.2 Culture in Organizations

Another aspect that contributes to the requisite variety of organizational systems is culture. Organizational culture comprises the attitudes, experiences, beliefs and values of people in an organization. It also embodies the organisation's interactional behaviour with its stakeholders. All organizations are run within certain cultural boundaries. As the definition of this culture is difficult and complex, its existence in organizations makes each organization different. Dahlbom and Mathiassen (1993) contend that, in order to understand a culture, one has to observe it in a practice, preferably by participating in that practice.

The information and knowledge that is required in organizations is always intertwined and embedded in a culture. Knowledge, like culture, is also interwoven in the same practice in which culture is observed. Practice is normally dependent on a definite situation. The fact that

knowledge is embedded in a practice, in the situations in which people perform that practice, makes the task of software developers very uncertain. In order to understand the situatedness of the organization, software developers should strive to acquire the operational knowledge of the organization. This knowledge can be gained by developers participating as workers or by means of formal and informal discussions with the workers and all other stakeholders in the organization. Without that, it is difficult to force the knowledge to be applicable to a definite situation such as an organizational information system.

Because of the need to capture organizational culture and knowledge in the organization, system development, hence software development may take several months or years to complete. In practice, therefore, and in the interest of time and project implementation schedules, many projects are continued before the developers have fully grasped the practices of the users.

Another problem in practice is the fact that many software practitioners develop software products for organizations in which they are not employed and have only limited knowledge of the organization's practices. The development of systems for these organizations without software developers having studied their practices forces them to chart knowledge in their heads, knowledge that is not applicable to the organization's culture. In consequence, they develop systems that depend on a practice that is not aligned to the current organization. The systems developed, therefore, portray a non-existent practice. To aggravate this problem, in practice, developers also rely heavily on the use of explicit procedures. As a result, bureaucratic or rule-based systems are developed.

In learning about a practice or when participating in a practice, people in organizations communicate using concepts. The role of concepts in organizations is discussed in Section 4.1.3 below.

4.1.3 The Role of Concepts in Information Systems Development (ISD)

“I know you believe you understood what you think I said, but I am not sure you realize what you heard is not what I meant.”

Pressman, 2000.

The above statement highlights one of the most misunderstood and neglected tasks in software development, that is, understanding the meaning of concepts used during

communication. These concepts may be business, domain or technical concepts. The development of concept negotiation techniques is a task that has not attracted enough effort in many software development approaches. Concepts are generally used as communication signs in organizations. The success of a development team in building a language community with all the stakeholders in the development process and, hence, in communicating concepts in the domain effectively, is regarded as one of the success metrics of software development. A language community is created when all the stakeholders involved can communicate and share their knowledge of a system with a common understanding. The use of concepts and their effective communication is of the utmost importance during the domain analysis stage. At this point, a study of the area to be serviced by the software product is carried out. This process requires system analysts to engage with business analysts, system architects and users so that the business environment can be established and understood before the process of system requirements gathering is commenced. Of course, this process also continues into the requirements-gathering stage.

Sowa (1976) characterizes a concept as an undefined primitive. In contrast to this, Buitelaar *et al.* (2003) argue that it is not always easy to identify a linguistic term as a concept. They contend that concepts do not exist or are not a given outside a specific domain. This rebuts Sowa's assertion that anything that a person can think of can be regarded as a concept. Aristotle, however, argued that concepts can be composed of other concepts that are broken down to a set of primitive elementary concepts. In this way, concepts can only be understood by enumerating their primitive elements.

In another discussion, Dahlbom and Mathiassen (1993) seem to agree with Buitelaar *et al.* (2003) that concepts are defined by people's practices. Concepts should satisfy three basic requirements: they should have an intensional aspect, a set of the concept's instances (its extension) and a set of linguistic realizations. Linguistic realizations refer to the multilingual terms that are applied to the concept (Buitelaar *et al.*, 2003). Concepts, therefore, should be defined in a particular context.

In fashioning systems and software products, it is important to note that these should be able to support people in the use and communication of concepts. As concepts are important communication tools, they are, therefore, the medium by which organizational information systems pass on information. These can be likened to obligatory passage points (Introna,

1997). Since each concept is defined within a domain, the role of context in assigning meaning to concepts should be discussed.

4.1.4 Context in Information Systems

As discussed above, concepts apply to a specific practice within a certain context. Context can be regarded as the environmental setting or the circumstances that dictate the occurrence of an event or the meaning associated with it. Furthermore, Ngwenyama and Lee (1997) believe the existence of organizational context serves as a framework for meaningful social actions. This was evidenced in their study of communicative messages, when the continual exchange of messages using emails among actors, eventually led to the building up and better understanding of context. Roque *et al.* (2003) stress the need for context to be understood if one is to successfully develop a software product artefact. They contend that the software artefact should be fashioned in such a way that it fits the context in which it will be used.

Also, organizational communication requires a language to transmit information between actors. In order to enable the actors to understand each other, this language should be entrenched within the context of the organization (Malinowski, 1923). Messages that are communicated in an organizational system are also situated in that particular context. Information systems should thus be tuned in such a way that they capture this organizational context. Capturing organizational context subsequently captures meaning, a very important facet needed to enable organizational actors to communicate.

These actors should also have “full knowledge of circumstances” if they are to have perfect explanations of the situations in which they find themselves (Roque *et al.*, 2003:111). However, it is most unlikely that any actor will gain complete understanding (or full knowledge) of a situation. The reason for this is that knowledge is usually modelled around some limiting factors that make knowledge worthwhile.

Context-building amongst organizational actors is a process of weaving together the different situational understandings of different actors, establishing threads of common understandings and of inter-subjective knowledge within the network (Goldkuhl, 2002; Dilley, 1999). Contrary to what many people think, context is not a static thing and is not a given. It is not self-evident in a situation but requires a constructive machinery to mould the varying situational meanings into a common understanding. Context, therefore, is an object of study that requires some analysis to arrive at an agreed and shared understanding. Furthermore, it is

within the shared meaning of some situatedness that the said context resides. Organizational context is ever changing and is continuous.

This dynamism in context poses a big challenge to software development if one has to capture the running context of an organization. Many software development processes lack methods for identifying, capturing and communicating context in the resultant software product. It is recommended that a methodology or a process be devised for building software and for changing it as and when the context changes. Without the presence of concepts that are communicated in a particular practice and without that practice being defined in a specific context, meaning cannot be communicated in any organizational system. Paradoxically, shared understanding or meaning is the basis for building a situational context. It is, therefore, of paramount importance that software development practices use a methodology or approach that captures this situational context.

Although it is agreed that software products should capture organizational context, there still remains a problem in the software development process of where, when and how this context can be captured, communicated and stored. Although currently, some available software development tools can capture the context, it is difficult to include this context in the software products developed.

This is compounded by the practice of software product development that has always been guided by the need to ‘*engineer*’ a product for use in industry. The use of the word engineering has restricted the focus on culture and context of organizations to such an extent that only those organizational aspects that can be formalized and, therefore, engineered, are implemented using software products. The impact of forcing the definition of engineering on to the software product development process is discussed in the next section.

4.2 The Practice of Software Engineering

This section highlights some problems to software development that have been caused by the use of the word and philosophy behind ‘engineering’. Several publications (Schach, 2005; Pressman, 2005; Heineman, 2000; IEEE, 1990) have given software engineering (SE) a plethora of definitions but all of them borrow their core meaning from the definition of the word “engineering”. The American Engineers Council for Professional Development (ECPD) defines engineering as:

“The creative application of scientific principles to design or develop structures, machines, apparatus, or manufacturing processes, or works utilizing them singly or in combination; or to construct or operate the same with full cognizance of their design; or to forecast their behaviour under specific operating conditions; all as respects an intended function, economics of operation and safety to life and property.”

Engineers Council for Professional Development (ECPD).

The underlining is for the researcher’s emphasis only. Focusing on the underlined phrases, we note that adherence to ‘*scientific principles*’ in software development has limited software developers’ attention to those aspects of software development that can be repeated and reproduced. This notion, therefore, disregards the softer elements of organizations such as culture and context that are usually tacit in nature and may not be repeatable. Coupled with the need to ‘*design*’ software artefacts, that is, to model and devise a representation that fits certain constraints, the process has actually limited the requisite variety of software products. In other words, design limits or restricts the variety of operating environments or of possible states in which a software product can be applied and used.

Current software development design processes have limited the ability of software products to be adaptable. This process forces software developers to concentrate on the ‘*intended function*’ of the software product. In organizations, however, such intended function is not a single, static or discrete function, but is dynamic, and continuously changing. In addition, the intended function has a running context that requires its capture and inclusion in the software product. There is, therefore, a limit to the extent to which developers can elicit the intended function of a living system such as an organization. The organization is considered as a living system on account of the fact that humans are always central to its existence.

Also in line with the ECPD definition of engineering, the IEEE Computer Society (1990) defined software engineering as the “*application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software*”. The underlined text is for the researcher’s emphasis only. As a model, Schach (2005:16) noted the need for developing “fault free software, delivered on time, within budget, and satisfying user’s needs”. As a framework, Pressman (2005:34) describes SE as “a process, a set of methods, and an array of tools”. Adding to that, Heinemann (2000) regards software engineering practice as a field aimed at developing

reliable and robust software products. All these definitions have had a very profound influence on the way software products are developed in practice. Their effect has been to curtail the inclusion of behavioural human aspects of organizations in software products.

Conford and Smithson (1996:11) added their voice by characterizing software engineering as a “*process of taking a specification and turning it into a software product*”. This specification emphasizes the functional requirements of a system. The way the functional specification is developed follows practices that are used in fields such as industrial manufacturing. If these practices are followed in the formulation of a software specification, the process will become very mechanistic (Pressman, 2005:45). The end product, the piece of software will neglect the behavioural characteristics of organizations.

It should also be noted that the scope of SE (Schach, 2005) is very wide and includes the study of mathematics, computer science, sociology, philosophy, economics, management and psychology to mention but a few disciplines. However, all these various fields that inform the practice of software engineering, SE has largely been confined to the hard sciences discipline, in which each software product is viewed as implementing a system with a specific goal, having a definite boundary and is closed and deterministic. Developers are trained to assume that for any specific input to a software system, there should also be a related predetermined specific output. This definitely contradicts the behaviour of human activity systems. In addition, it is worrying that most software engineering practitioners are barely familiar with more than one of the disciplines mentioned above, especially the psycho-cognitive disciplines such as psychology. This alone is a disadvantage to the software development profession.

Current software engineering practices disregard the fact that the software product is not an end but is rather a means to an end: the implementation of organizational information systems. As a means to an end, during its development, the software product should take cognizance of the characteristics of organizational systems as supported by these three theories: the theory of organized activity (TOA), actor-network theory (ANT) and activity theory (AT). This also supports the argument that information systems have multiple goals and are non-deterministic and that their boundaries are not definitive but are permeable and open (Du Plooy, 2004). In fact, the discipline of software engineering should be regarded as a soft discipline in the fashioning of which Checkland’s (1999) soft systems-thinking methodologies can play a major role.

It is important to see how the definition of software engineering has contributed to the way developers engage in software engineering practices. At the same time, the view given to organizations, their information systems and culture should be described in relation to the problems encountered in the development of software products. Organizations exhibit a dynamic attribute that cannot be represented by a static piece of software. As described above, there is a paradox in the practice of software development, with a dynamic environment being forced onto a static mode of representation. This paradox may safely be described as the software development problem.

4.2.1 The Software Development Problem

The software development problem was coined a ‘*crisis*’ at the first NATO software engineering conference (Randell, n.d.). The issues discussed during this conference included project budgetary, quality and timing problems (Baker, 2006). As Brooks (1987) stated, the field of software development is characterized by projects that miss their schedules, exhaust their budgets before completion, and, finally, result in flawed software products. These issues are barely managed and satisfied during software development. Although many attempts have been done to address these problems, the software development problem still persists to this day.

In his contribution to the software development debate, Mullet (1999) cites two reasons for software development process continuing to be in a crisis. He attributes these to the lack of relevant software engineering (SE) education and the fact that SE is regarded as a craft. It is argued that many developers are concerned with the final product and not with the process itself. This problem is exacerbated by the separation of the software development and system development processes. Software developers should not separate the process of developing software from that of building organizational information systems, he added. Instead, the software development process should be regarded as integral part of the system development process.

According to Whitten *et al.* (2004), a system development process includes a set of activities, methods, best practices, deliverables and automated tools that developers use to develop and maintain information systems. Like the software development process a system development process, therefore, requires a system development methodology. In discussing system development methodology, Whitten *et al.* (2004) included the maintainability aspects of both the software products and the resultant information systems in the system development

process. Ensuring that the maintainability of the two is considered during development can also help to reduce the problems in both software and organizational information systems.

In another memoir, Basden (2001) decried the continual lack of return on investment from information systems investments. He tracked down these problems to software products and their characteristics which subsequently emerge in the final information system. He found that these problems originated during the developmental stages of these information systems. Pressman (2005) noted that the software product always carries the responsibility of delivering other products, such as information systems and their outputs. This piece of software, therefore, plays a dual role in its life cycle (Pressman, 2005). This dual role also puts an extra burden on developers, who have to ensure that the right product is developed before it is used in information systems.

Basden (2001) contended that there was something very wrong with the way in which the software product as a system artefact is developed. It is, therefore, very important to find and resolve the problems that lead to the development of flawed pieces of software. Another problem that has bedevilled the development of software products is that software developers have always construed the process of requirements analysis (from which the specification is built) as a process of eliciting plans and formal definition of procedures that should be enforced in an information system. This emphasis on formalization has resulted in neglect of the softer issues inherent in organizational information systems. In the end, the formal plans and procedures are expected to run the IS just as efficiently as the actual organizational system. It is pointed out that these formal plans often lack the business and domain aspects of the organizational system and that they only capture the formal system specifications that can be technically represented.

Furthermore, issues considered during the software development process play a vital role in shaping and determining the quality of the software product. It should be borne in mind that lack of quality is one of the issues that contributed to the software crises. Basden (2001) also noted four areas of concern that could be addressed to improve the quality of the software product. The first is on how the software artefact is fashioned for use. The second looks at the development of technology which we use to fashion the software artefact. The third focuses on how the software artefact is used and, finally, the focus is on the users' and developers' overall perspectives on the use of technology. This last aspect requires people to consider the social construction nature of the software product. Basden's concerns can only be addressed

by finding a software development methodology that integrates these four requirements into a software process model. These can be in addition to the focus on the design of reusable components and on the innovative elements of a software product design. These innovative elements of a software product represent the domain-related additions that make the difference between domain packages. These contribute to the life-responsiveness of the information system.

With regard to domain, De Oliveira *et al.* (2006) regard the lack of domain knowledge by software developers as the greatest problem in the development process. Although the processes of requirements elicitation and knowledge-gathering are very laborious, knowledge sharing and reuse of knowledge in software development is however, very limited. In these processes, one needs to explain the same concepts repetitively to different software development personnel. Developers have to study and learn the domain while simultaneously linking this to the tasks to which it relates. These tasks pertain to the problem domain that has to be addressed by the software product (De Oliveira *et al.*, 2006).

Brooks (1987) grouped the difficulties inherent in the software development process into three categories that is, the fundamental nature, the difficulties intrinsic in the nature of software and the accidents. He defines the fundamental nature of a software product as a formalized construct of interlocking concepts, sets of data, and associations among data items, algorithms, and invocations of roles. He noted that the explication of this essence or fundamental nature poses a big challenge in software development. The other two are much easier to identify. The next section discusses some of the development practices that have been used to define the essence of a software product.

4.3 Software Product Development Practices

In this section, various practices being employed by practitioners in order to improve the quality of software products are discussed. As mentioned earlier, most methods are aimed at improving the production rate, quality and maintainability issues of the software products. Software kernels and software product lines are the two practices that will be discussed in this section.

A software kernel is a part of software system that is relatively stable over time and is used in a family of related products (Dittrich & Sestoft, 2005). The basic requirement of a kernel is to provide and encapsulate core functionality for the software product. It is the foundation upon

which other successive software products can be built or from which they can evolve. The purpose of a software kernel is to increase the reusability, production rate and, possibly, the consistency of production of software products. On the other hand, the software product line approach (SPLA) to software development can help developers to find the functionality, structure of the kernel, and also the structure that a kernel imposes on the applications based on it (Dittrich & Sestoft, 2005).

The Carnegie Mellon Software Engineering Institute (CMSEI, n.d.) defines a software product line (SPL) as a “ set of software-intensive systems that share a common, managed set of features satisfying the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way.”

The problem with the software product line approach is that, it is a predictive type of software reuse strategy and does not allow the time-dependent adaptation of the software product to the environment and context of the organization. This approach focuses on software artefacts that can only be packaged if there is a possibility that they may be used in one or more products that are in the same product line (Krueger, n.d.). SPL focuses on the creation of a portfolio of software artefacts that comprise a set of features shared in common. These are packaged together and stored. When reused, the software products can decrease the time to market, reduce costs and facilitate the evolution of software pieces.

However, the SPL development strategy is ineffective in providing flexibility, fast response times, capturing of semantics and context or in providing capabilities for mass customizations (CMSEI, n.d.). SPL and software kernels have not had much success as development paradigms because they neither support nor encourage the development of adaptive systems. One of the major reasons for the lack of success of SPL and the kernel approach is that they do not allow a holistic transfer of analysis model characteristics, that is, the business, domain and specification models through to the design and, subsequently, to the implementation model. Because of this, they persist in neglecting the behavioural characteristics of organizations in their implementation strategies. The next section will discuss some issues of communication in software development.

4.3.1 Communication in Software Development

The failure of several software development processes or methodologies to transfer the full analysis model to lower levels of design and implementation can be attributed partly to poor

communication techniques in these process models. The communication is in two parts, namely, during requirements gathering and during the transfer of the requirements into the analysis model through to the implementation model. This section will consider both communication issues and, finally, highlights the major communication problems that are faced by software developers.

In all software development processes, the analysis phase is tasked with creating a true reflection of the environment of the organization. For a successful software development project, communication with users has to be given high priority in most of the developmental phases, including analysis. Harris and Weistroffer (2009), in particular, investigated several empirical studies and found a positive correlation between user involvement and system development success. They noted that users are more important during the preliminary phases of feasibility studies, requirements determination, and of the later stages of the design of input and output documents and the installation of the systems (Harris and Weistroffer, 2009). Lehtola *et al.* (2009) proposed the creation of solution concepts, a solution package that could be used to guide requirements engineering activities. Their proposal is based on the need to have a strong functional link between the business strategy and the software development process.

Garcia-Duque *et al.* (2009) further proposed another method for improving processes of gathering requirements, analysing them and later revising them. Their intention was to increase the fidelity to the business system requirements of the requirements gathered. They noted that this could be achieved through a continuous process of analysis, verification and revision. On the need to include the human context in software products, Fuentes-Fernandez and Gomez-Sanz (2009) provided a framework based on socio-psychological activity theory that could guide the gathering of social aspects of systems and, subsequently, include them in the software products. In addition, they saw a greater need for developing analytical tools that could be used to elicit the human context of business systems. All these endeavours aim at addressing the communication problem in software development. The communication problem is not limited to the analysis stage alone but extends to all phases of software development. The basic requirement is to address the collection of socio-technical systems requirements (Bryl *et al.*, 2009).

The analysis phase should produce a model that is descriptive and has the form of a computationally independent model (CIM) (Aßmann *et al.*, 2006). The CIM possesses as

little platform-dependent information as possible, at the same time ensuring that the customers' viewpoint is maintained. This customer can be regarded to as the business user. This computationally independent viewpoint captures the environment and requirements of the system. Many techniques have been used in traditional analysis modelling to ensure that this analysis model is expressed in terms of the problem domain. The intended relationship in communication that should exist between the analysis, design and the implementation models is illustrated in Figure 4.1.

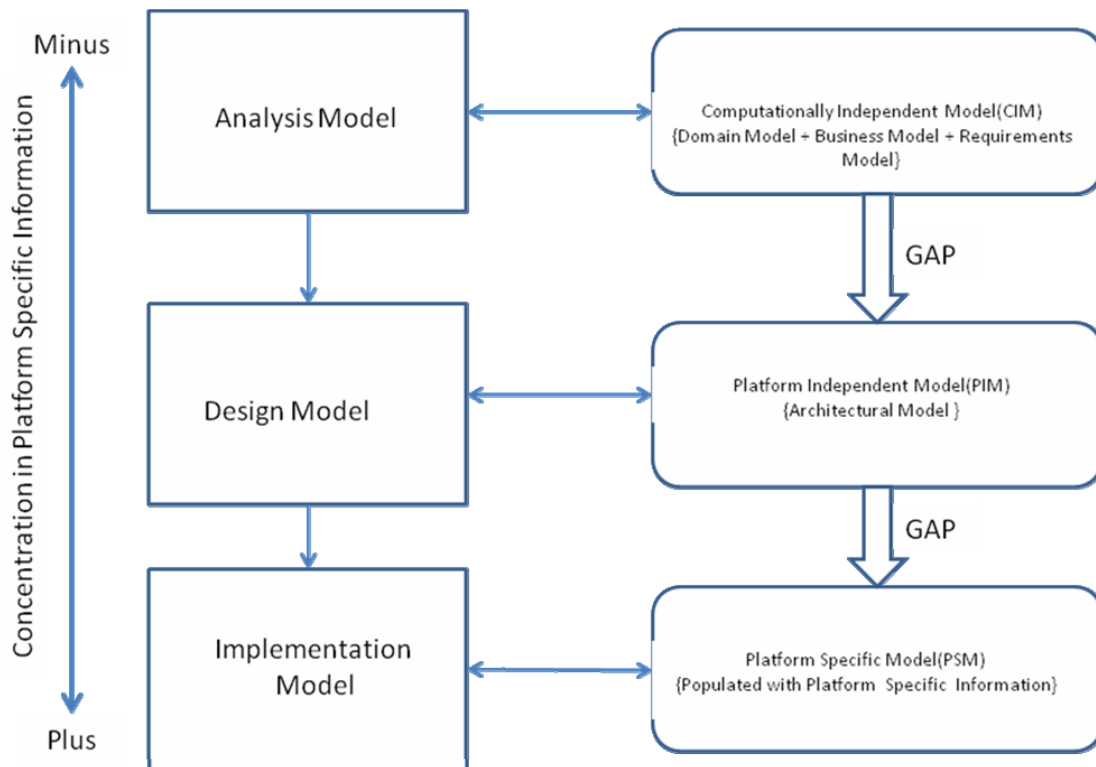


Figure 4.1: Communication Gap in Software Development

As discussed earlier, the analysis model is derived from the environment of the organizational system. This environment is characterized by the domain and business information, both of which are context related. In addition, the system requirements, which derive their fit from the organizational context, are also added. The purpose of the analysis model is to capture the triplet: domain model, business model and the requirements model. The requirements model manifests itself as the system specification. Also, the domain model is a product of the process of domain engineering. Bjørner (2008) lists understanding and capturing human behaviour as the prime purpose of this process. It should be noted that organizational human behaviour which comprise a bigger part of the domain model has to be described formally or informally and communicated throughout all the software developmental stages. This

domain model should capture the concepts used in the domain field as well as the relationships between these concepts. Bjørner (2008) also lists intrinsic, supporting technologies, management and organization, rules and regulations, scripts and human behaviour as the guiding principles for developing domain models.

The business model is tasked with capturing the organization's rules of business. Lastly, the requirements model, which models the system specification, is tasked with capturing the functional and non-functional system requirements (Aßmann *et al.*, 2006). On the other hand, the design model is the architectural model of the system and, at this stage; it should capture the system from the designer's viewpoint but should still be platform-independent. Lastly, the implementation model is gradually populated with platform-specific details as discussed by Aßmann *et al.* (2006). There is a gap between the different types of domain knowledge in the domain model and the substance and form of software artefacts that are constructed. To reduce this, Falbo *et al.* (2002) proposed an infrastructure specification that, with its semantics as captured by the domain model, could be used as input to the implementation phase of the software development process. Wand and Weber (1990) advocated a complete translation of the analytic model attributes through to the implementation model.

Software developers usually talk of formal system specifications. These generally refer to program code, thus implying that the specifications are platform-dependent. For specifications to be platform-independent, some informal specification methods should be found and used. Bjørner (2008) argues that this may require the specification to be drawn up in some sort of natural language. This has not been the case with all specifications, especially when developers arrive at the later stages of development. Bjørner (2008:62) decries the fact that "there are very many aspects of requirements that we today, 2008, do not know how to capture formally..." As a result, many analyses and design specifications do not represent the true world view of a system.

The insistence of developers on formalization results in the development of mechanistic development methods. Because of this formalization in these mechanistic development methods, from the analysis model through to the implementation model, the platform-specific information has been allowed to creep into the system. The major problem faced by software developers is that of translating all the characteristics of the analysis model (CIM), as shown in Figure 4.1, to the implementation model through the design model (PIM). This is normally because, at the end of the analysis stage, the system requirements are translated to a

specification model (SM). This SM is an instantiation of parts of the functions of the system. It focuses on those system aspects that can be formalized and the informal aspects are ignored.

As stated by Aßmann *et al.* (2006:257), “*a specification model is a prescriptive model, representing a set of artefacts by a set of concepts, their interrelations, and constraints under the closed world assumption*”. The failure of the SM to transfer descriptive information captured by the analysis model to subsequent stages poses a serious problem in software development. In another attempt to improve communication in software development practice, Gunter *et al.* (2000) developed a reference model that could be used to bridge the gap between the analysis and design phases of software development and increase communication.

4. 3.2 Software Engineering Reference Model

The software engineering reference model developed by Gunter *et al.* (2000) consists of five basic components: domain knowledge, user requirements, software specification, software program and the programming platform. These are illustrated in Figure 4.2. This model is supported by Aßmann *et al.*'s. (2006) description of domain knowledge, in which they state that the domain knowledge, together with user requirements is used to describe the environment. On the other hand, the system to be developed is described by the program and the programming platform, the environment and the system being linked through the software specification. The software specification or model as portrayed here is a product of both the analysis model and the design model.

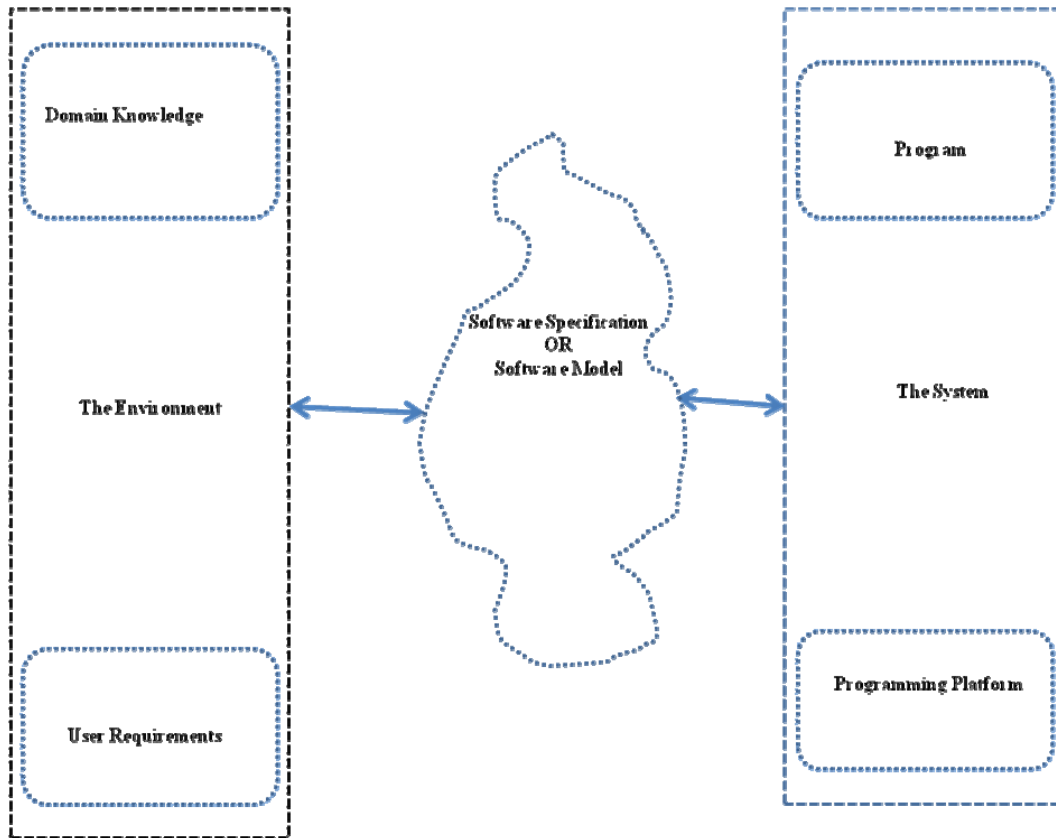


Figure 4.2: Software Reference Model (Adapted from Gunter *et al.*, 2000)

This reference model (Gunter *et al.*, 2000) highlights the need for a good communication medium between the environment, the system, and the medium. In other words, there needs to be a mapping between the environment and the system, in this case, the system specification or software model that maintains the fidelity of system characteristics in the environment to the developed system. Gunter *et al.* (2000:54) concluded that a framework for “classifying and relating key software artefacts” is needed. The framework can be used to connect the formalization needed in artefact development to software engineering practice. In fact, Gunter *et al.* (2000:54) proposed a “systematic methodology for formal software engineering”. This systematic methodology should be able to transfer all the possible system states in the environment to the system implemented. Although this reference model includes domain knowledge and user requirements in its discussion, it does not include the relationship between the analysis model as discussed in section 4.3.1 and the design and implementation models. At the end, it falls short of capturing the softer aspects of organizations in software products.

Another communication technique that has survived the test of time is the database conceptual schema. This schema has been used to mediate between the environment (user

interface), the applications (programs) and the storage facilities (databases) in almost all the systems developed to date. In the next section it will be argued that the way this schema concept has been implemented also greatly affects the way systems work in real life.

4.4 The Need for a Conceptual Schema

The conceptual schema was proposed in the 1970s by the American National Standards Institute (ANSI). The schema has been used for encoding knowledge in information systems (Sowa, n.d.). It captures and stores application knowledge of information systems (Sowa, 2000). The schema and the role it plays in information systems are illustrated in Figure 4.3 below.

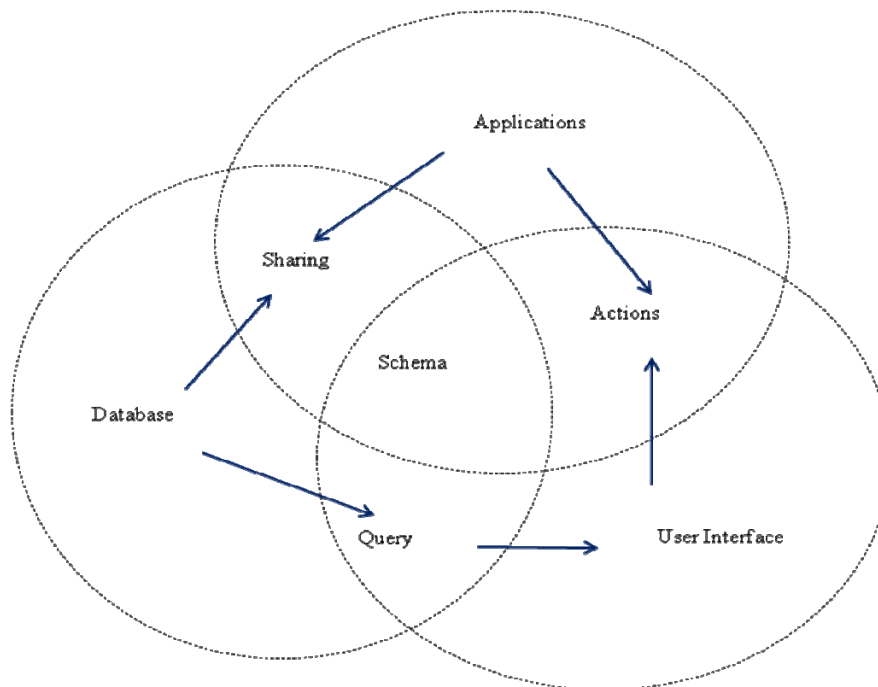


Figure 4.3: Components of an Integrated Information System (Adapted from Sowa, 2000)

As depicted in Figure 4.3, the user interface is the bridge between the system and users. The user interface is populated with facilities that allow it to access the database. By means of this user interface, users can query and edit contents of the database. At the same time, the interface facilities should be able to perform actions and provide services for the application programs.

Between the database and the applications, the database provides the data needed to run the application programs. This database allows data sharing and also provides a permanent

storage facility of data for the information system. Lastly, the application programs are the coded processes and activities that are required in any system to perform any prescribed task.

As the communication engine and the brains behind information systems (IS), the schema should provide common knowledge, application entities and relationships that exist in the organizational system. In a real world scenario, the duties performed or allocated to a schema are usually the responsibility of a human actor. Although human actors allow syntactic subscription of actions to perform any given task, they also use concepts in ill-defined contextual situations. In contrast, replacement of a human mediator by a schema that communicates at a syntactic logical level only reduces the requisite variety of the subsequent information system. Practically, the schema concept allows for an effective and efficient system design but is, however, very syntactic and is not adaptive. The major reason for the schema being syntactic is that it was developed and fashioned to interact with and fit to an already established database query language, the popular Structured Query Language (SQL). Sowa (1976) tried to circumvent the dictates of the schema by using conceptual graphs in querying and communication. However, this attempt has not yielded tangible benefits and until now, the system development fraternity is still subject to the mechanistic dictates of this schema. This section concentrated on some pertinent issues that remain unaddressed in software and system development. These issues continue to have a negative influence on the way in which software products are developed. Section 4.5 can be viewed as the epistemological and ontological grounding of organizational information systems and, hence, of the software products.

4.5 The Mechanistic Nature of Information Systems

“There is a reason why computers have not yet become fervent natural language speakers. (It’s not a matter of processing power and never will be): we simply are not programming them correctly.”

El Baze 2005.

El Baze’s (2005) comment points to the fact that, although developers can perfectly implement whatever they design, software products fail because it is the design itself that is grounded on improper assumptions. Software product designs do not take cognizance of the central role played by people in organizational information systems. With this in mind El Baze (2005) laments that “we simply are not programming them correctly”.

In addition, the syntactic nature of software products is also the result of the philosophical assumptions that guide the development of software products. Embedded in these philosophies are the understandings given to a system, information system, systematicity and system formation (Gasche, 1986) as well as to the concept of system engineering, as discussed in Section 4.2 above.

Another reason is that current practices in information systems development are strongly based on a mechanistic world view. Based fundamentally on the functionalist paradigm, this view regards the world as being ordered, rational and unchanging. Monod (2007) strongly blames IS practitioners' unquestionable and faithful adoption and use of this rationality principle that make them believe that the world can be reduced to discrete functional units that can be represented as rules and algorithms. This world view also supposes that the knowledge and information that people have about the world can also be defined explicitly. In contrast to this notion, the knowledge and information used by people in organizations is mostly tacit and intuitive and cannot be identified explicitly and fed into rationalistic rules and algorithms. It is, therefore, important to motivate for a software development paradigm that rejects extreme rationalism and technological determinism as is currently the case and accepts that the world view is voluntarist, chaotic and subject to interpretation. This is the relativistic stance.

The use of the schema concept at the centre of the information systems (IS) to mediate between the user interface, database and applications has dictated the way in which software products are developed. Also most software development philosophies have been guided by the need for any system to accommodate the SQL as the query language, hence their strict adherence to the schema concept.

By definition, information systems consist of people, machines and processes that are required to accomplish specific tasks. As part of information systems, computing machines are clearly products of mechanistic thinking. Instead of limiting this mechanistic thinking philosophy to the development of machines, developers have also applied this mechanistic principle to the way people and processes work in organizations. This mechanistic way of thinking depends on the idea of a bureaucracy, of an explicitly formalized organization working only with explicit data. The mechanistic modelling of systems removes truth in the real world from the resultant model. This is called prescriptive modelling and can be

contrasted to descriptive modelling, in which the truth in reality has to be captured in reality (Aßmann *et al.*, 2006).

As stated earlier, the software product development process requires developers to exhaustively capture all the possible scenarios that can be assumed about a system. This, however, is difficult as it is not possible to capture and match the requirements of organizations that always have a running context that determines the next step of action in the software product. The failure to capture the running context results in the development of closed and predictable systems (Avison & Fitzgerald, 1995).

In addition to this, many organizational systems have databases that are populated with data objects that “ignore physical objects, processes, people and their intentions” (Sowa, 2000:56). It should be noted that computer systems do not have intentions of their own (Oinas-Kukkonen and Harjumaa, 2009). Rather, those who build and distribute the technology should inscribe the assumed intentions into the computer systems. These computers systems can be grouped into autogenous technologies (that shape the behaviour and attitude of those using them), endogenous technologies (technologies developed with user voluntariness toward attitude or behaviour change) and lastly, exogenous technologies (technologies that provide the means to personalize the assigned goals). The most important issue is that this role should be given to software developers and, most importantly, to the methodologies they use.

In addition, the databases pay very limited attention to the semantic content needed in information systems development. Its greatest disadvantage is that the technology that people have entrusted to run their information systems cannot deduce relationships between signs as can the human mind. In short, software development has followed an engineering, mechanistic approach that assumes that the tasks and steps to be performed by a software product can be predetermined and fully specified (Brown *et al.*, 2004). This is a rational approach that is in line with developers' rational view of the world. These rationalists believe in the “power of good representations to predict and control social change” (Hirschheim *et al.*, 1995:3). To improve the usefulness of the software systems, developers need to combine and evenly balance a mechanistic understanding of computing machines with a romantic appreciation of the complexity of people, social organizations and information use. There must be a balance between the structural and functional views given to information systems.

This brief background to the mechanistic nature of information systems will lead us to the discussion of specific philosophies that are a foundation of these aspects.

4.5.1 Mechanistic Systems, Systematicity and System-Formation

According to the Theory of Organised Complexity (TOC) (Checkland, 1999:78), systems in general exhibit a general hierarchy of levels in which each level is more complex than the level below it. Each such higher level also has emergent properties that are not found at lower levels. It should be noted that these emergent properties are the result of system formation, in which the whole exhibits characteristics that cannot be found in the individual sub-systems that combine to form it. As noted by Checkland (1999:78), “*neither a one level epistemology nor a one level ontology is possible*” to describe the sum total of the subsystems. This expresses the concept that, in a hierarchy of systems forming the whole, each level has different distinct epistemological and ontological views. In other words, the views of the lower levels of the system can never be the same as those of the whole system. Put in another way, the behavioural characteristics of a subsystem, when added together will generate a system whole that has behavioural characteristics completely different from those of its constituents. The interaction of the components creates some emergent properties that are a by-product of the interaction and these manifest themselves in the whole as new characteristics. This principle supports the notion that an aggregation of mechanistic components cannot have the same properties as the whole. This also brings in the two notions of systematicity and system formation. In other words, systematicity that led to the mechanistic development of systems looks at the extent to which a system can be regarded as an ordered, hierarchical arrangement of components. Its focus is the machine and the artificial versus the preservation of the natural that is encompassed in the romantic world view. System formation in turn looks at the ordered, organised building up of the whole system from its components.

In explaining these two terms, it is important to define the concept ‘*whole*’ as a synthesis or unity of parts. These parts are so close that their activities and interactions are affected by the closeness of these parts as a result of the synthesis. Information systems can be taken as a subset of all the possible states that make up the whole system. Such a view posits information system infrastructures as artefacts that are not discrete pieces of components or fragmentary constructions. Instead information systems should be conceived as “artefacts made up of a continuous connection; chained arrangement of its constitutive parts. These

constitutive parts, if fragmented and reassembled, one cannot come up with the original whole” (Gasche 1986:5-7) (*sic*). This contrasts with Aristotle’s maxim that the whole is equal to the sum of its constitutive parts. When the whole is broken up, it loses its requisite variety by negating some other possible states. At the same time, when the system is reassembled from the parts, the theory of organized complexity explains the introduction of emergent properties that never existed in the original whole. This now constitutes a paradox, where one cannot holistically get an organizational whole from the products of its deconstruction, that is, the constitutive parts.

In its totality, a general system that is reconstituted (developed) from the mechanistic principles of reductionism cannot have that unity of purpose or focus and/or the horizon of meaning, sense and context that gives it the attributes of a total or a whole system (Gasche 1986).

Although, through the application of reductionism, systematicity and system-formation have been used to construct general systems (Gasche 1986:8), the system components could not be reunited into a “one, well rounded-off system”. Gasche (1986) sums it up by asserting that, although reductionism is a very good condition of successful idealization of information systems development, the resultant information system itself lacks idealization (Gasche 1986). In relation to the principles of systematicity and system formation, there is also another principle, that of romanticism.

4.5.1.1 Romanticism

Romanticism in systems can be described using Gasche’s (1986) notion of anti-systematic thought. This is related to the anti-positivist notion described in Chapter 2. Romanticism argues for the negation of systematicity and system-formation while, at the same time, allowing the concept of “the fragmentary ...” and the process of deconstruction of the greatest totality to be based on the former. According to Gasche (1986:7), deconstruction concerns itself with finding the limits of the acceptance of “systematicity and system formation”. Deconstruction is a conceptualization process of what constitutes the so-called “general system”. As Gasche (1986:7-8) noted: “the general system is not the universal essence of systematicity; rather it represents the ordered cluster of traits of possibilities which, in one and the same movement, constitute and deconstitute systems.”

The paradox that faces system developers emanates from the fact that one cannot tackle a system development project without breaking it up into manageable chunks. On the other hand, the whole cannot be reconstituted from these chunks. A development approach that reduces the gap between these two poles, one that is able to reintroduce the romanticism that existed in the original system in the developed systems, has thus to be found. This section supports the need for a software development methodology that allows the deconstruction of original systems to the constitutive parts and then allows the reconstruction without losing the softer human aspects.

By assuming the romantic world view, IS developers consider a holistic view and an acceptance of the organizational system, where culture and social context play a part in the execution of a task. In the eyes of romanticists, “processes and change” are at the forefront of system “contemplation, understanding, interpretation and feeling” in contrast to the structures and systems of the mechanistic world view (Dahlbom and Mathiassen, 1997:501). In this view, change in organizational systems is taken as “unpredictable and beyond human control, the expression of hidden and unknowable forces” (Hirschheim *et al.*, 1995:3).

During system development, developers need to consider the structural features of systems that should be conceived in any development process if the system is to achieve its goal. As noted by Wyssusek (2004:4303), at this juncture, information systems analysis and design “should not solely be guided by an instrumental-technical and / or a practical-hermeneutical cognitive interest but also an emancipatory interest...” In this case, developers can, therefore, overcome their long “self inflicted cognitive constraints” (Wyssusek, 2004:4303). Many information systems analysis and design techniques take the ontological position that neglects the modelling of softer human issues in organizations. The process of structurally breaking up the system features during analysis and design, and later recombining them during implementation so as to recreate the whole, poses serious limitations to the current reductionist processes in many system development approaches.

4.5.2 Transition to Romantic Systems

Problem-solving in the interpretive and neo-humanist paradigms cannot be guided by syntactic rules and formalizations as in the functionalist paradigm. On that note, IS developers are urged not to concentrate on tractability and objectivity in this field. Any emphasis on these two aspects distances and isolates the resultant artefact from complexities of everyday social changes. This problem is prevalent during the analysis phase, during

which requirements are gathered, and is aggravated during the transition from analysis to design of the systems. At the implementation stage, the people aspect of organizational IS is almost completely forgotten.

Yu (1997) noted that many requirements-modelling techniques focus on the completeness, consistency and also the automation of system requirements verification: neglecting the early requirements tasks that are stakeholder oriented. These are tasked with checking “how the intended system would meet original goals” (Yu, 1997:1), including, but not limited to, why and how the system will address stakeholder needs. It is more important to leverage the requirements engineering effort on the reasons why requirements are needed and gathered than on the specification of what the system must do (Yu, 1997). In fact, as Roque *et al.* (2003) put it, there is more to *requirements than elicitation*. Requirements are an everyday social construction, in which human and non-human actors participate. This position requires information system (IS) developers to look holistically at the organizational context. The organizational context includes information technology (IT), the people and procedures in the organization and the interaction of these three actors. The mechanistic nature of systems running in organizations is the result of looking at only a single part of the system, the *servicing system*. The servicing system consists of the IT infrastructure only. The *served system*, which consists of people in the organization, is generally neglected (Kawalek & Leonard, 1996). The constituents of the served system are lost during the requirements gathering stage, and the few that are captured during this stage are also lost during the transition to the design stage.

Current system development methodologies typically force many organizational systems to embed their business rules, organizational culture, practice and their human aspect in the technological side of IS. It is, therefore, necessary to motivate for developmental methods that liberate the human aspects of the organization from the bondage of technology. Kawalek and Leonard (1996) motivate for a winning configuration in which the human and technological aspects are evenly balanced.

In many systems, current modelling techniques, in which systematicity and system formation are profound, the task approach described earlier is used. As Keen (1988) noted this approach encourages an IS culture of thin technical orientation. He then argued for a role-based approach in which the IS and user relationship is evenly balanced. The task approach works efficiently in an environment in which discrete processes can be used. However, the IS

environment is not that discrete. In support of Keen (1988), Lehman (1994) described a domain that is served by software applications as continuous, irregular and dynamic. In contrast, the software products that serve this domain are themselves distinct, bounded and, unless human decision and action has changed them, are static. Software products have long been used as the technological artefacts that map the social organizational system to information systems. It is, therefore, necessary to focus any attempt to reduce the mechanistic nature of information systems on the software product. The software product is derived from the software model (Gunter *et al.*, 2000) as a software specification; hence the software model becomes the blueprint for the software product.

The software model as a piece of organizational representation and also as the blueprint for the final software product is viewed by Lehman (1994) as an estimate, essentially incomplete artefact with entrenched assumptions. This should not be surprising since, as already discussed, the software model is a product of the mechanistic development paradigm. More often than not, it lacks the domain and business model aspects of organizations. The next section looks at three theoretical frameworks that can be used to appraise organizational systems.

4.6 Activity Theory, Actor-Network Theory and Theory of Organized Action

This section uses the three philosophical theoretical frameworks: Activity Theory (AT), Actor-Network Theory (ANT) and Theory of Organized Action (TOA), to look at organizational systems. The problems caused by the reductionist tendencies of systematicity can be viewed through the principle of human activity theory as a lens. Human activity theory is derived from the principles of activity theory.

4.6.1 Activity Theory

AT framework has been used for a very long time now to explain and understand human activity systems. An activity is described as the smallest indivisible, action-oriented and goal-directed process that can be found in a system. Taking an activity as a basic unit of analysis in organizations, we use AT to explain the “coherence of individual actions in a larger social context” (Roque *et al.*, 2003:112). An information system (IS) is considered as an assembly of individual activities that work in synergy to accomplish an organizational goal. These activities are the ones upon which the task approach of IS development are based.

As activities make up the IS, each activity in turn is composed of actors. An activity can be described as an organized assembly of actors that work together to accomplish a stated goal. The activity is then “driven by peoples’ needs” (Fuentes-Fernandez and Gomez-Sanz, 2009:3) and as a process, it accepts objects as input and the output should be able to satisfy peoples’ needs. An aggregation of the activities, therefore, constitutes an information system. The actors in an activity can be humans, technological artefacts (hardware and software) or a combination thereof.

These basic components of information systems have to communicate through a mediator in order to satisfy organizational requirements. The mediators are also responsible for acting as communication channels between the actors. Although they are actors, software products are also mediators in the system. They, therefore, mediate between individual humans, between humans and technological artefacts (hardware and software) and between purely technological artefacts. This state of affairs, in which technology and people coexist, leads to the understanding that mediation software products work in a socio-technical human activity system.

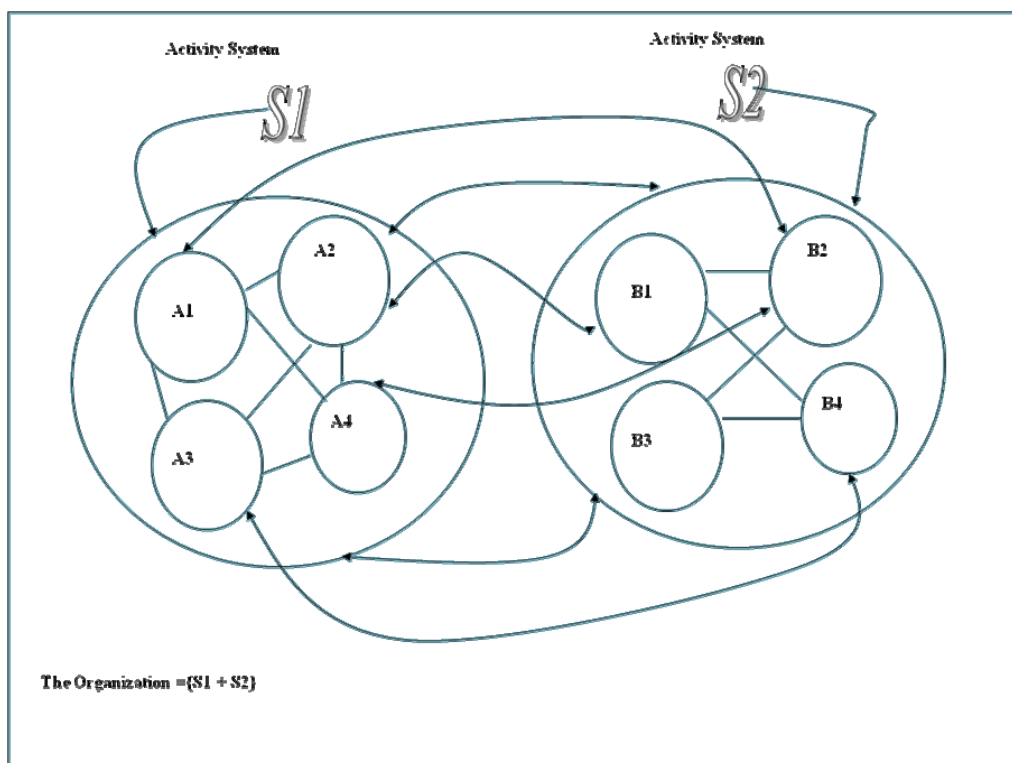


Figure 4.4: The Concept of an Activity System (Adapted from Roque et al., 2003)

Figure 4.4 shows A1 - A4 and B1 - B4 as individual activities and activity system S1- {A1 - A4} and activity system S2- {B1 - B4} are different activity systems. Besides being activities, A1 - A4 and B1 - B4 are also referred to as actors. S1 + S2 give rise to yet another combined activity system: the organizational system.

In practice, the goals for activities A1 - A4 and B1 - B4 can be determined *a priori* but it is difficult to determine the goals which result from their interaction to make activity systems S1 and S2 respectively. This can further be explained using Checkland (1999)'s theory of organized complexity described in Section 4.5.1 above. In addition, the combination of S1 and S2 to make "O", the whole organizational system, becomes even more complicated. This section will be used to appraise organizational information systems using activity theory as a lens. The human activity theory can be used to explain the non-deterministic and complex behaviour of information systems.

As discussed earlier, the current software development methods have been designed to follow the Aristotelian principle that the sum of the parts is equal to the whole. Owing to the inclusion of people as actors in these systems, these methods have failed to consider the human behavioural aspect of activity systems. In the behavioural sciences, the whole may not necessarily be equal to the sum of its parts. On that note, determination of the nature of a process (activity) and its outcomes and "ignoring changes in motives and goals, ignoring actors, human and non-human, ignoring the multiplicity of disciplinary agencies involved" have been the reason why information systems often fail (Roque *et al.*, 2003:113). The software product as the mediator should be fashioned so as to accommodate all these contingencies. AT can, at least, be used to explain the gap that exists between organizational IS and IS products.

4.6.2 The Social Network Aspect of Activity Systems

As already discussed and with reference to Figure 4.4, each activity system A1 - A4, B1 - B4, S1 and S2 can be referred to as actors. Within each activity, a software product mediates and creates communication channels. Since each activity system consists of a network of interacting actors, each activity system may, therefore, be referred to as an actor-network. In this regard, Latour's (1999) Actor-Network Theory (ANT) can be used here as a theoretical lens to explain the social interactions and relationships that exist between each actor in the activity system.

ANT regards each component of an activity system as an actor that is influenced by other actors through the mediation of these actors. The actor, as an “author of inscriptions”, is a network itself and centre of translations. Each actor is also influenced by the relationships established by itself as a node in the network in which other actors participate (Roque *et al.*, 2003:113).

Because of the presence of humans in IS, the actors participating in the IS as an activity system exhibit both human voluntarism and technological determinism whose interplay results in the emergence of complex social characteristics. In the spirit of ANT, any actor loaded with either inscriptions or translations, whether a mediator or the mediated, should be fashioned in such a way that it is allowed to evolve and co-evolve with other actors. It should be allowed to adapt to the ever-changing requirements of the environment. The environment in which these processes take place is not static but dynamic, continuously changing and adapting to new demands. The question, therefore, remains as to how to make the software product capable of accommodating these responsibilities.

4.6.3 Human Activity Systems (HAS)

Human beings have never been predictable. Their behaviour is always changing. Suchman (1987) contended that it is very difficult, if not impossible, to consider all contingencies in an *a priori* prescription of a human action. Rationalization of human actions *a priori* or *a posteriori* overlooks much detail that is situated in the running context, such as the detail taken during a course of action in everyday life (Roque *et al.*, 2003).

Roque *et al.* (2003) urge developers to guard against neglecting variability and independence during the development of IS products. They add that reliance on procedural and functional descriptions of organizations and individual roles as complete accounts of the social dynamics in an organization should be discouraged (Roque *et al.*, 2003). In mapping organizational dynamics, software development should not neglect the contextual settings of the system. Contextual information is, however, tacit and is shared among actors in the same organization. Software developers have always neglected the contextual characteristics of information systems (IS) because of their tacit attributes. Up to now little research has been done to incorporate context and intuition into the development of information systems (Beynon *et al.*, 2008). Another theory, the theory of organized activity (TOA) which is related to activity system theory, is discussed in Section 4.6.4 below.

4.6.4 Theory of Organized Activity (TOA).

Like the human activity system (HAS), Holt (1997) developed the theory of organized activity (TOA) that is also based on human (organized) activities. An organized activity is a dependent variable of the social interaction of people in a particular setting. Looking at an IS as an activity system, Cordeiro and Filipe (n.d.) view the technical aspect of IS as playing a supporting role to the organizational human activity.

They also describe the human action, that is, the action performed by a human actor in this actor-network as comprising interests and actors. These interests, together with the actors, are, therefore, responsible for the actions. While humans can have interests, technical machines cannot have interests and, therefore, cannot be assigned any organizational responsibilities. The end result is that the technical aspects of IS cannot perform actions (Cordeiro & Filipe, n.d.).

In short, although technological artefacts may be components of an organized activity, their failure to inscribe interests places them at a disadvantage if they are to be assigned any responsibilities for an action. The development of a software product as a technical artefact should, therefore, include ways that allow them to be assigned responsibilities. We can refer to this notion as humanizing the technical artefact, in this case the software product. The software development approaches that are employed in the development of the software product should be revisited and adjusted so as to allow the incorporation of methods that will capture the human and social nature of the activity systems.

These three theories, AT, ANT and TOA, discuss and support a very important characteristic of organizations and their information systems. In reflecting on software development, these theories urge researchers and developers of IS products to consider organizational culture, the context in which the software products will be used, how to capture the tacit information in organizations, as well as the non-determinism of socio-technical construction systems such as organizations. As discussed earlier, these three theories can be used as a basis for a conceptual grounding of romantic information systems. A conceptual framework for romantic information systems can be described as:

a framework that allows the development of socially-constructed systems that capture and maintain the softer elements of organizational systems such as culture, social context, and

semantics and to a certain extent pragmatics. These systems must be adaptive, dynamic, evolvable and innovative.

The whole idea is to bring some intuition, tacit information and meaning into the software product. This characterization may seem farfetched, but, existing literature like Weber (2003), Hohmann (2007), Yu (1995, 1997), Beynon *et al* (2008) and Soffer *et al.* (2001), to mention but a few, are already calling for a need of a romantic framework like the one proposed in this study.

Having motivated for a new way of developing software products in previous sections, in the next section the researcher looks at the approach to, or the methodology requirements of, a software development process.

4.7 The Approach versus Methodology Debate

The purpose of this study is to motivate for a new software development approach. As such, this section delves into the fundamental tenets that are considered as differentiating between an approach to and a methodology for software development. Starting with an approach, Checkland (1999) refers to an approach as a way of going about tackling a problem while Iivari *et al.* (1998:166) regard it as a “class of methodologies which share the fundamental concepts and principles for ISD”. Such an approach may not, however, be very prescriptive regarding the method to be used when tackling a problem and may not be required to provide methodology instances. This implies that, an approach may be formulated without any accompanying methodologies (Iivari *et al.*, 1998).

Brown *et al.* (2004) consider an approach as the lighter form of a process. They add that, unlike methods, an approach is less prescriptive than a method and that adapting it to specific scenarios is fairly simple. Furthermore, an approach, by its very nature, can be developed into one or more specific methodologies. In the field of software development, Roque *et al.* (2003) characterize software development (SD) approaches as classes of methods that map areas of similar methodological thinking. Approaches look at methods that share goals, guiding principles, fundamental concepts and principles for the SD process. These approaches are derived from a specific paradigm, for example, structured, object-oriented and agile approaches. Software development approaches or methodologies are developed by method engineers (Gonzalez-Perez & Henderson-Sellers, 2006). In developing these approaches method engineers use concepts in the method domain, that later need to be “instantiated by software developers when they apply the methodology” or the approach.

In software development, methodology is defined by Schach (2005) as the science of methods. Pressman (2005) further characterizes a software development methodology as a framework that can be used to structure, plan, and control the process of developing software products. He goes on to state that the methodological framework consists of the two elements below:

- A software development philosophy: this is the approach to the software development process; and
- Some tools, models and methods that can assist in the development of software process.

Pressman's definition is likely to mean the same thing as Roque *et al.* (2003) and Iivari *et al.*'s (1998) characterization of an approach. Hence it shall be regarded as such. A methodology may also be regarded as a formalized approach or as a combination of steps and deliverables that are needed for the development of a software product (Dennis *et al.*, 2001). This, however, takes place at a lower level of the hierarchy than the approach discussed above. In contrast to an approach, although a methodology also includes the philosophical underpinning found in an approach, Benson and Standing (2005:203) also list "phases, procedures, rules, techniques, tools, documentation management and training for developers" as components of a methodology.

Characteristically different, but somehow similar, Bjørner (2008:41) defines a methodology as a "study of and knowledge about methods" which according to Gonzalez-Perez and Henderson-Sellers (2006) comprises of methods, techniques and tools. To avoid confusing methodology with method, Bjørner (2008:41) goes on to describe a method as "a set of principles for selecting amongst, and applying, a set of designated techniques and tools that allow analysis and construction of artefacts". In this he supports Benson and Standing (2005) but also emphasizes the prescriptiveness of methods, techniques and tools that are used in a software development methodology.

In turn, Bjørner (2008:41) describes a principle as "an accepted or professed rule of action or conduct..." He also regards a technique as a "specific procedure, routine or approach that characterizes a technical skill" and views a tool as "an instrument for performing mechanical operations, a person used by another for his own ends... to work or shape with a tool" Bjørner (2008:41).

A term that requires further elucidation is the word “paradigm”. Schach (2005:25) characterizes a paradigm as a style of software development. This is a way of doing something as contrasted to the actual meaning of “a model or pattern”. Of importance is the fact that paradigms are not disjoint but may overlap in terms of approaches and methodologies. It should be accepted that different researchers and authors have different definitions for approach and methodology, including the methods. This is further reflected in Table 4.1 below which contains a list of some of these definitions from different authors.

These excerpts are however, not exhaustive. There is much literature that discusses these three subjects in different expert disciplines and, therefore, readers are urged to acknowledge the varied contextual interpretations given to them. The definitions in Table 4.1 are, however, derived from authors and researchers in the field of software development and information systems. This makes them relevant to the discussion in this chapter.

Table 4.1: Characterizing Approach, Methodology and Method

Characterizing Approach, Methodology and Method	
Concept	Concept Characterization
Approach	An approach is a set of goals, guiding principles, fundamental concepts, and principles for the system development process that drives interpretations and actions in system development (Iivari <i>et al.</i> , 1998).
Methodology	A methodology is defined as an explicit way of structuring one's own thinking and action. It contains models and reflects a particular perspective of reality based on different philosophical paradigms. A methodology implies a time-dependent sequence of thinking and action stages (Jayaratna, 1990).
Methodology	A methodology is a set of recommended steps, approaches, rules, processes, documents, control procedures, methods, techniques and tools for developers, which covers a whole life cycle of an information system (Repa, 2004).
Methodology	A methodology is a collection of procedures, techniques, tools and documentation aids which will help the systems developers in their efforts to implement a new information system. A methodology will consist of phases, themselves consisting of sub phases, which will guide the systems developers in their choice of the techniques that might be appropriate at each stage of the project and also help them plan, manage, control and evaluate information systems projects (Avison & Fitzgerald, 2006).
Methodology	A methodology can be interpreted as an organized collection of concepts, methods, beliefs, values and normative principles supported by material resources (Hirschheim <i>et al.</i> , 1995).
Method	A method is regarded as a path or a procedure by which the developer proceeds from a problem of a certain class to a solution of a certain class. The steps of a method impose some ordering on the decisions to be taken during development (Jackson, 1981).
Method	A method defines what should be done in a particular phase of the Information System (IS) development process. A method is always based on a particular approach (functional, data, object-oriented). A method always covers just part(s) of the IS development process or some point of view such as data, function, hardware (Repa, 2004).

The discussion in Table 4.1 positions an approach at a higher level of abstraction, descriptiveness and flexibility than does the methodology. The most important distinction between these two is the fact that methodology is more prescriptive, that the steps to be followed may be too limiting and are clearly defined. In an approach, a myriad of methodologies can be formulated. The most important aspect of a methodology as discussed by Jayaratma (1990) is the fact that the philosophical thinking in a methodology is a time-dependent sequence of activities which also accommodate some thinking, principles and normative principles (Hirschheim *et al.*, 1995). In light of that, Truex *et al.* (2000) warn methodology engineers to realise that most of the software development activities may not fit in the methodological frame that they are accustomed to, especially at very low process levels. This a-methodological frame may necessitate the omission of these activities in a

methodology layout. This is despite the fact that they encourage methodology engineers to clearly state the notation and guidelines to the use of their methodologies. A closer look at Table 4.1 shows that a method is derived from a methodology, in other words the degree of prescriptiveness increases as well.

Referring back to approaches, there are several ways of categorizing these. Some may use the dichotomy between hard and soft systems approaches and some may use the paradigmatic distinction encompassed in the methodology such as structured, object-oriented and agile approaches. In this research a paradigmatic classification according to how one perceives the world view has been used to categorize them under the realistic or relativistic paradigms. (*See Section 6.4.*) Using this classification, three categories are found, that is, the traditional or structured, approaches in transition and behavioural approaches. (*See Figure 6.1.*) The approaches using the methodological process followed during software development are discussed in Section 4.8 below.

4.8 Software Development Approaches

According to Roque *et al.* (2003), software development (SD) approaches lie between software development (SD) paradigms and SD methods. However, methodology, approach and paradigm apply to the whole process of software development. Currently, three SD approaches are found: the structured approach, the object-oriented (OO) approach and the agile approach. The discussion below focuses on the basic tenets that these approaches address. These tenets focus on the philosophical thinking, that is, on the beliefs and values that are enshrined in the approach.

4.8.1 Structured Development Approach

The structured development approach can be divided into process-oriented and data-centred approaches. Techniques such as the functional decomposition diagrams (FDD), data flow diagrams (DFD), entity relationship diagrams (ERD) and data structure diagrams (DSD) have been employed to model processes and data in this approach.

The structured approach is a reductionist type of approach which assumes that a system can be broken down systematically into a hierarchy of functions (processes) and sub-functions (sub-processes). This approach also uses deductive reasoning (moving from the general to the specific) (Brown *et al.*, 2004). The hierarchy theory discussed by Checkland (1999) can be used to explain the philosophical grounding of the structured development approach. The

hierarchy theory stipulates that systems can be regarded as having different levels of complexity. This difference in complexity is noted between levels either at a higher or lower aggregation level of the system. In short, it accepts that systems can be broken down into simpler subsystems and that the whole can be formed from an aggregation of its parts. In between levels, however, there are fundamental differences in their structure, complexity and composition.

The structured development approach is well grounded in the principles of systematicity and system formation, as explained in Section 4.5.1. It concentrates on analysis and algorithms but pays very little attention to synthesis or to relationships between parts during the process of software development. Several software-development methodologies based on the classical waterfall process model and some modifications have been used within the confines of this structured approach. The next section will discuss the agile approaches.

4.8.2 Object-Oriented Approach (OOA)

The item of consideration in the object-oriented approach (OOA) is the object. The object is designed in such a way as to encapsulate both data and procedures, unlike structured approaches which treat these two separately. System behaviour is an attempt to look at synthesis and the relationships depend on the interaction between the objects of the system. Objects sharing common characteristics form a class and all the different classes found in a domain area form the system. This contrasts with the hierarchical decomposition of functions found in structured approaches. Relationships between classes are considered and this adds to the semantic richness of the system to be developed (Brown *et al.*, 2004). However, the fundamental philosophical groundings in OOA are not very different from those of the structured approaches. Paradigmatically, both rely on the functionalist paradigm and the realist stance.

4.8.3 Agile Approaches

Agile approaches borrow their philosophical grounding from the theory of adaptive systems (Highsmith, 2004). Systems should be agile and adaptable to the environment in which they operate. A host of methods summarized in Brown *et al.* (2004) have been used in this line of thinking. The major differences between agile approaches and the traditional (structured and OO approaches) that are important to the development of information systems are summarized in Table 4.2 below.

Table 4.2: Philosophical Groundings of Software Development Approaches (Adapted from Brown et al., 2004: 4139)

Philosophical Groundings of Software Development Approaches		
	Traditional Approaches	Agile Approaches
Fundamental Philosophical Assumptions	Project proceeds in linear fashion. Deterministic approach that eliminates uncertainty through reasoning.	Feature-driven and proceeds in an iterative, evolutionary manner. Ambiguity and uncertainty are solved through cycles of rapid feedback and continuous improvement.
Process Model	Waterfall, spiral, etc. They emphasize linear sequence of process steps.	Evolutionary development model proposed by Gilb (1985).
Systems Thinking	Hard systems.	Hard and soft systems.
Thought Process	Driven by strict, predetermined rules established by the process model, which basically makes one react in predictable ways to unusual situations that might arise.	Urges developers to use patterns to solve problems by relying on their ability to innovate, depending on the contingency.
Dealing with Complexity	Assumes that complexity and ambiguity can be predicted, measured and corrected.	Deals with complexity and ambiguity by using the ingenuity of people and relies on rapid feedback and adaptability.
Customer Involvement	Not directly involved in the development process.	Mandatory, active participation throughout the development process.
Team Composition	Relatively homogenous.	Self-organizing teams involving relevant stakeholders who may have diverse perspectives and disparate goals.

It is worth noting that Brown *et al.* (2004) classified agile approaches as operating in the neo-humanist paradigm. In practice, however, these approaches seem to operate in both the functionalist and humanist paradigms. This is supported in Table 6.3 by the many incidences of techniques that are used in both paradigms, as well as by its classification under the paradigms in transition. (See Section 6.4.). Although some of the agile techniques fall within the humanist paradigm, the problem they face is that, after capturing the human attributes

inherent in organizational systems, they lack a process for including and maintaining these characteristics in the software products.

It can be seen that these current software development approaches place a greater premium on processes and technologies than on people and their behaviour, a notion that is strictly grounded in hard systems development methodologies. Roque *et al.* (2003), as proponents of a new approach to software products development, proposed some goals that should be taken into consideration if the software products are to capture the romanticism inherent in organizational natural settings. These goals have to be grounded within a certain philosophical underpinning for them to satisfy the development approach.

4.8.4 Goals of a Software Development Approach

It is accepted that each software-development process requires guiding principles, methods, tasks and activities that are guided by a specific body of knowledge. By the same norm, the software development approach in turn dictates the structure and form of the software development process to be followed. A software development process, life cycle or software process can be conceived as a structure imposed on the development of a software product.

Accordingly, each development approach should at least satisfy the following goals in addition to many other paradigmatic requirements. As discussed by Roque *et al.* (2003), some of the goals mentioned here require an approach that would:

- Allow software developers to frame the SD activities, supported by the relation between context and mediators of the activities that mould that context;
- Achieve an understanding of the SD activities on the proposed framework, viewing IS development as a social-technical phenomenon within cultural and historical envelope;
- Deal explicitly with contextuality in SD as the key to performing emancipatory movements; and
- Capture semantics, pragmatics and context in the software product.

As emancipators, software developers are able to anticipate users' needs and are thus able to produce artefacts that have the capability to capture third-party understandings of a situation.

These goals for software development approaches led to the discussion of requirements for developing romantic software products.

4.9 Model Requirements for a Romantic Software Product

This discussion enforces some requirements to any approach that is intended to yield romantic software products. The requirements are derived as a summary of the existing literature in the field of software development that has been discussed in this chapter. These requirements include but are not limited to the following:

- The software model should be sufficiently broad and flexible to accommodate a variety of possibilities.
- The software model should be easily actionable but should also allow loose coupling between applications and data sources.
- The software model should capture the context of the situation. This will ensure the capturing of all possible life states of the system.
- The software model should be sufficiently broad (abstract) to allow it to adapt to various and ever-changing sets of problems. It should allow for a dynamic transition between possible states of the system depending on the context.
- The software model should also be deep enough to provide a rich, detailed and valuable context.

Since the software model is tasked with the transcription of possible system states to the software product, it should have some characteristics that meet these requirements. With this in mind, the development of romantic software products should be guided by a framework that possesses sufficient obligatory passage and translation points (Introna, 1997; Mavetera, 2004b). These obligatory passage and translation points will allow the mediation of different contexts for different organizational software products. The issues raised in this chapter that can guide work on the development of romantic software products are summarized in the next section. In line with these issues, a software development approach that will ensure the development of romantic software products should encompass all or most of these requirements.

4.10 Software Development Issues

Reflecting on the discussion of this chapter, we discover that, from the beginning, software development approaches have been grounded on philosophical frameworks that use the realist world view as their cornerstone. This world view, in conjunction with the functionalist paradigm, views the served system separately from the serving system. Throughout this chapter, a consistent argument has been followed that motivates for a shift from the mechanistic software product-development approaches currently in use to one that allows the development of romantic software products. This approach is grounded in the relativistic world view and the interpretive neo-humanist paradigm. In moving to this relativistic stance, the following issues have to be considered:

- There should be a switch from hard systems approaches to soft system approaches.
- The methodological approach should have a way of capturing the dynamic nature of the ever-running context in organizations. On this note, the approach should ensure that software developers are able to study organizational environment and live in it so that they can have a situated practice and experience this practice before they embark on any software development project.
- The approach should ensure a transition from a task-based approach to the role-based approach.
- Developers should be able to build a language community with all stakeholders, that is, there should be a linguistic model that could be used to negotiate a shared understanding of the concepts found in a system. This requirement supports the need for improved communication methods, techniques and tools that can be used during the development process.
- In the current modern and pervasive computing environments, in which software development is outsourced offshore, the development approach should have a platform to enable different developers to share their understanding of the system requirements regardless of their location.
- The software product should be able to capture semantic and pragmatic (tacit) information in organizations.

- The social construction nature of information systems, as expounded by the three theories, AT, ANT and TOA, should be considered and given the highest priority.
- In addition to being able to model behaviour, the organizational culture and context should also be captured, as these are always an integral part of organizational behaviour.
- The functional requirements and the system requirements should be mapped from the organizational environment to the systems platform through a software model that does not neglect the social or human aspects of the organizational system.
- De Oliveira *et al.* (2006) state that a common repository, a guiding framework to the software process, domain and task knowledge are prerequisites for a sound environment for software development and should be captured in a software development environment (SDE). This repository is a store for all information related to the software development life cycle (SDLC). In addition, each software development process requires knowledge about the organization. This knowledge sets the context of the software product.
- Data and experience gained from previous software projects, in addition to identifying relevant key personnel that can work on a project should also be acquired, stored and reused.
- An analysis model, derived from the domain theory, should be designed for a family of systems in the same domain.

Lastly, this approach should completely ignore the reductionist tendencies of the structured, object-oriented and partly agile development approaches that currently guide our practice as software practitioners. The software development process needs to adopt the soft systems approach. More importantly, the methods used should be adapted to enable them to capture, store and maintain organizational behaviour in the software product. In conclusion, as this chapter has portrayed, a behavioural, socio-technical approach to software development should be the foundation for the development of romantic software products.

4.11 Summary

This chapter explored the softer issues of organizational information systems that are usually not considered in the hard sciences discipline such as computer science and engineering. To clarify the need for paradigm change, several theoretical frameworks were discussed, such as the theory of organized complexity, activity theory, actor-network theory and the theory of organized activity. These theories, together with the softer elements of organizations such as culture, context and people's behaviour, motivate for software development approaches that are grounded in the relativistic, interpretive or neo-humanist paradigms and that are able to capture, store and share with all stakeholders the knowledge found in organizations.

The discussion also highlighted the requirements of a software development approach. In this context an approach is at a higher abstraction level than a methodology. More so, an approach should be descriptive rather than prescribe methods to users, a task that is delegated to a methodology. A methodology, however, should comprise methods, techniques and tools that are used in a software development process. The content of this chapter will be used in Chapter 6 as additional and supplementary input to the data collected from software practitioners. It is, therefore, important for the reader to grasp that, although there is a myriad of attributes that can characterize organizations, solutions to capturing those described in this chapter have persistently eluded software practitioners. These will be at the focus of our proposed solution set. This discussion should be read in conjunction with Chapter 5, so as to enable the reader to understand the role that these two chapters play in this study. The conceptual grounding that supports the use of ontologies in software development will be discussed in Chapter 5.

CHAPTER FIVE

HISTORY AND NATURE OF ONTOLOGY

5.0 Introduction

The discussion in Chapter 4 highlighted many contentious issues that are still grey areas in the software development field. Among the problems still facing developers is the need to maintain the characteristics of the business and domain models of systems, from the analysis stage to the design and implementation stages (*See Section 4.3.1*). Furthermore, it has been stated that, in order to avoid prescriptiveness, the requirements specification model should be captured and documented in a domain-related language.

This chapter introduces the information systems ontology concept, an artefact that has been undergoing continuous development for roughly the past two decades or so. The term ontology has been used in IS in the early 1990s by Tom Gruber and his research team (Neches *et al.*, 1991; Gruber, 1993), when he was working at Stanford University. Its use in computer science, information systems (IS) and information technology (IT) has been increasing gradually. In recent times it has been used in the fields of Semantic Web and database systems (Gruber, 2008), library sciences (Ding & Foo, 2002), routing systems (Winter & Tomko, 2006), information systems and software development (Mavetera, 2007; Aßmann *et al.*, 2006; Dristas *et al.*, 2005; Corcho *et al.*, 2006; Wand & Weber, 1993; 2002), to mention but a few. Ontology has also been used to access legacy resources (Simonov *et al.*, 2004) through an ontology-driven natural language web-based access system. More interestingly, Soffer *et al.* (2001) used ontologies to bridge the gap between business requirements and off-the-shelf information systems capabilities in order to adapt the business to the software capabilities. As is evident from these few examples, there are many fields that already use ontologies and software development is but one specific application area.

The purpose of this discussion is to outline explicitly the characteristics of ontologies that can be used in software development and, most importantly, to support the objective of this thesis of developing a methodological approach to software development that is ontology-driven. It is hypothesized that an ontological approach to software development can only be achieved if the current gap in traditional software approaches is closed by introducing ontologies in the software development process. The problems highlighted in the traditional approaches are

mainly the result of the lack of approaches that can address the cultural, contextual and finally, the humanist requirements of information systems, as discussed in Chapter 4.

As discussed earlier, this chapter, which focuses on the nature, characteristics and application areas of ontologies, will build a theoretical grounding that will be used to motivate the inclusion of ontologies in software development. Their inclusion may occur at several levels, such as the development or runtime stages of the software products. It may also occur at the knowledge base or process levels (Haller & Oren, 2006). This discussion starts by looking briefly at the history of ontology from its philosophical definition in metaphysics and its subsequent incorporation in information systems. After discussion of the characteristics of ontologies, the various types of these ontologies applicable to this research are also discussed.

To provide a good grounding for its use in the development of software products, the discussion continues by focusing on the methodological and architectural ontology perspective. Finally, a working definition of ontology is discussed, together with that of an ontology-driven framework of components that can be adopted for use in software development. This framework, together with the information systems issues discussed in chapter 4 is used in Chapter 6 to develop an ontology-driven software development approach.

5.1 What is Ontology? A Brief Description

“Ontology makes knowledge visible and accessible and enables teams to share their knowledge and profit from experience.”

Sheryl Torr-Brown (In-PharmaTechnologist.com, 2005)

Ontology is a word that originated from classical philosophy as a branch of meta-physics. It is referred to as the science of being (lower-case ‘b’) (Ruiz & Hilera, 2006) or as the study of existence (Hacking, 2002). As a study of essence, ontology started as a way of categorizing things and establishing the nature of their existence (Corcho *et al.*, 2006). It also deals with issues such as how people perceive the world and with general issues of the nature of things as opposed to specific theories about particular things. Checkland (1999) holds that it is a concept that deals with the nature of the world or with what it contains. This definition does not look at the individual fragments of existence but at the general. As Hacking (2002:2) argues, ontology constitutes the thought study of “What there is.” It should be noted that this term, ‘Ontology’ written with a capital letter ‘O’ has an uncountable reading (Guarino, 1998). A typical example is the statement ‘*Ontology is the study of existence*’ which, in this context,

refers to a specific discipline of study. The philosophical ontology is “neither reducible to, nor identical with language or its formalism” (Zúñiga, 2001:188). However, the language can be used to describe this ontology.

In its use in information systems, Guarino (1998) describes the concept “ontology” (with a lower-case ‘o’) as having a countable reading. As such, people can refer to ontologies if they talk of more than one type (kind) of a thing. In the context described by Hacking (2002:2), this looks at what is there to individuate and characterize, in other words at, “the particulars that fall under them”. An example of such a countable reading is the statement:

Surgical ontology and pharmaceutical ontology are both examples of medical ontologies.

Hacking (2002:1) stresses the need to contrast the study of ‘being’ (lower-case ‘b’) with the ‘Being’ (upper case ‘B’) that is used to individuate entities. In addition, Hacking (2002:2) suggested that people should think of ontology in general as the “What is there that we can individuate?” This view includes both physical and abstract concepts, such as classes, types of entities and even conceptual ideas that people may have about a thing. Ruiz and Hilera (2006) regard even mathematical objects and imaginary things as ‘Beings’, regardless of whether they are fictitious or unreal. This type of ontology takes as its epistemological stance the notion that objects may be both physical and abstract at the same time or may be natural or created by the action of man (the artificial).

Adding to this, Corcho *et al.* (2006:4) distinguished between ‘an ontology’, which is a classification of things, and ‘Ontology,’ which is a branch of philosophy. They argue that while the philosophical ‘ontology’ may not be computer-processable, the ontology used in information systems should be “codified in a machine interpretable language, that is, the human perspective of ontology should be changed to the computer perspective”.

Therefore, ontology (with a lower-case ‘o’) can also be viewed as an engineering artefact, as a product of design science (Hevner *et al.*, 2004; Peffers *et al.*, 2008). Guarino (1998) contends that, in its engineering form, ontology consists of a specific vocabulary used to describe a certain reality or what Shanks *et al.* (2003:56) called “theories about the structure and behaviour of the real world in general”. These are contrasted to subject specific ontologies such as domain and process ontologies. The vocabulary used is accompanied by a set of explicit assumptions, which give the intended meaning of such a vocabulary. In it, certain assumptions have to be made that govern the ontology in its application domain.

These assumptions, however, are expressed in the form of first-order logical theory. The vocabulary contains binary and unary predicate names, which are called relations and concepts respectively (Sowa, 2000).

In systems development, ontology is, therefore, a “representation of knowledge based on objects, concepts and entities existing within the studied area, as well as the relationships existing among them” (Ruiz & Hilera, 2006:50). The purpose of ontologies is to provide machine processable semantic information that can be shared by organizational actors such as humans and software agents (Simonov *et al.*, 2004; Hofferer, 2007).

In-PharmaTechnologist.com (2005) further characterizes ontology as a branch of applied science that deals with the development and use of knowledge networks. They also regard it as a science of context and communication. This brief on the essence of ontology enables us to appreciate the different application areas in which ontologies can be used.

5.2 Background to Ontology Applications

Ontologies have found various application areas in the modelling of static knowledge and the semantic Web, in which they define, carry and share knowledge on the Web (Corcho *et al.*, 2006:2). In fact, ontology is used in almost all scientific disciplines (Haller & Oren, 2006; Taniar & Rahayu, 2006; Calero, Ruiz & Piattini, 2006) and thus much research is being done in this field. The scientific disciplines in which ontologies have found some application include knowledge engineering, information engineering, agent-based information systems, knowledge management and artificial intelligence, to name but a few.

Many knowledge-based problems found in information management application, whether intra-, inter- or extra-organizational, can be exploited using ontologies. These activities, as noted in In-PharmaTechnologist.com (2005), include enhanced information retrieval, text mining, annotation of databases (Gruber, 2008), data-mining and the development of tools to facilitate data-discovery and decision-making. According to Ruiz and Hilera (2006:51), in software development, ontologies assist in avoiding “problems and errors at all stages of the software product life cycle”. Of particular importance is their role at the requirements analysis stage, where they carry the domain knowledge of the discipline into the software product itself and subsequently to the maintenance stage. Ruiz and Hilera (2006) add that, at the maintenance stage, ontologies enhance the understanding of requests for modification and of the maintained system.

5.3 Other Views of Ontology

In addition to Guarino’s (1998) definition, Gruber (1993) and Studer (1998) describe ontology as an explicit formal specification of a shared conceptualization. Formalization in the ontology deals with machine readability and conformance (syntactics) to specific standards (Mavetera, 2004b). Explicit specification incorporates the clear identification of concepts, their properties and relations, the functions, constraints and axioms (semantics) within a universe of discourse. Explicitness, therefore, refers to the clarity of its existence and meaning to all people involved in the subject matter.

In discussing this further, it should be noted that, if something is clear to a subject, it should be “sense-making” to that subject (Mavetera, 2004b). The addition of the phrase “shared conceptualization” implies the existence of at least two subjects who share a common inter-subjective world view. This brings in the notions of consensus and agreement on the use of concepts that make up the ontology and their meaning. Furthermore, there should be some sense of mutual understanding of the concepts among people, as well as of how they are applied in that same contextual environment (pragmatics).

To explain the concept of shared conceptualization further, Figure 5.1 shows actors A and B, both of whom have different world views.

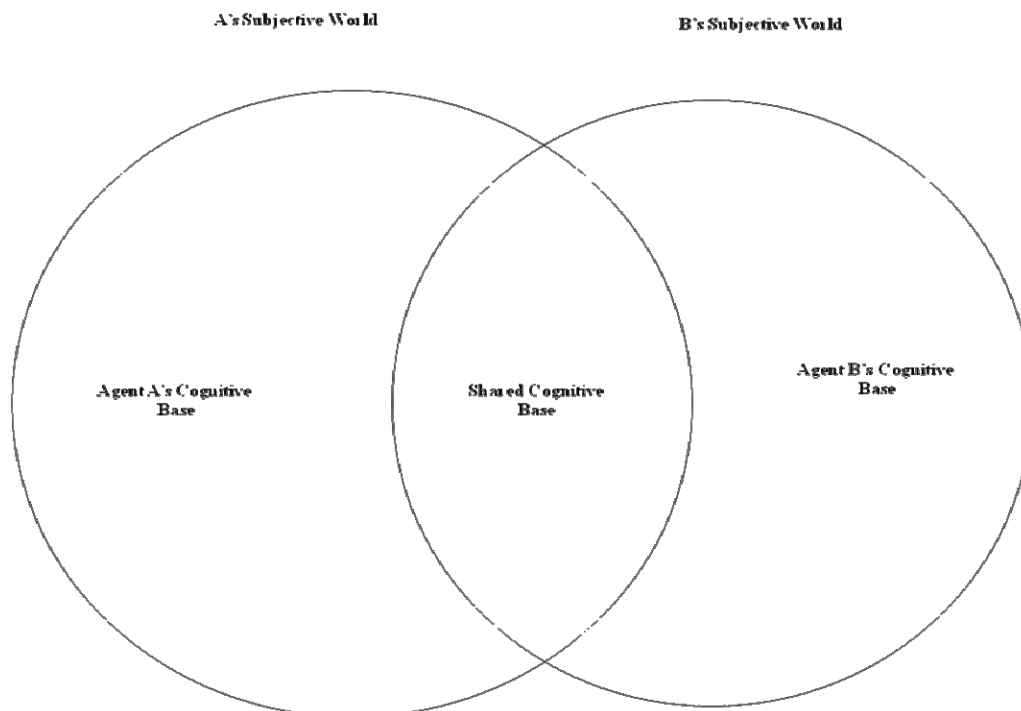


Figure 5.1: Different Realms of the World (Adapted from Goldkuhl, 2002)

Figure 5.1 shows that each actor possesses its own view of the world and that this world view can be likened to the actor's ontology. The world view resides in the actors' individual cognitive base. Actors A and B should, therefore, have a shared understanding of their cognitive bases if they are to communicate effectively. This shared cognitive base is depicted as the intersection of the cognitive bases of actors A and B, as illustrated in Figure 5.1. To summarize, the shared cognitive base is, therefore, synonymous with the shared conceptualization of a topic area that is heavily dependent on the context of the situation. This shared cognitive base resembles the ontology of the two actors in a particular domain. This ontology characteristic is very important in organizations when attempts are made to capture organizational context.

At a grammar level, Neches *et al.* (1991) and Uschold and Gruninger (2004) define ontology as the basic terms and relations comprising the vocabulary of a topic area, as well as the rules for combining the terms and their relations so as to define extensions to this vocabulary. The meanings of these terms should (ideally) be grounded in some form of logic. In agreement with this definition, Swartout *et al.* (1997) and Aßmann *et al.* (2006) also describe ontologies as groups or collections of hierarchically structured concepts that are abstracted from a certain domain. This definition views ontologies as groups of facts that can be used as a skeletal foundation for a knowledge base (Mavetera 2004b), that is, it is a set of representational terms in the universe of discourse. This argument equates ontologies to a group of concepts as defined by Dahlbom and Mathiassen (1995) but not as packages of information.

In another discussion, Dristas *et al.* (2005:6) describe ontology as an “attempt to express an exhaustive conceptual scheme within a given domain”. They argue that ontology can be used as a good basis for information modelling. In other words, it is an artefact that is well placed to capture the information content of an organization. Thus, contrary to the earlier discussion that does not accept ontologies as being packages of information, the use of ontologies in information modelling can be used to support Mavetera's (2004b) thesis that ontologies can be viewed as information packets. However, Aßmann *et al.* (2006) advocate the view and use of ontologies as models. In this form, ontology becomes a model shared by a group of people in a certain field. Unlike ordinary models, ontologies need to be shared and should adhere to the open world assumption principle. The open world assumption principle holds that

anything that is not explicitly expressed by ontology is not refuted or negated but is regarded as a fact that is not available and, therefore, not known.

In utilizing the ontology artefact, different researchers have varied perspectives, depending on the proposed outcomes. There are, however, two prominent perspectives that are currently gaining ground in information systems: the methodological and the architectural views of the ontology artefact.

5.4 Ontology Perspectives

The methodological perspective focuses on a very high interdisciplinary approach. In this approach, the role of philosophy and linguistics in analyzing the structure of reality is discussed, the intention being to get a clear and unambiguous vocabulary that is used in a particular domain. This thesis does not discuss this perspective in great detail but focuses on the architectural or structural perspective of ontologies.

5.4.1 Architectural Characteristics of Ontology

Architecturally, ontologies consist of classes, relations, axioms and instances (Neches *et al.*, 1991; Aßmann *et al.*, 2006). These ontology classes represent concepts in a domain area (Corcho *et al.*, 2006) and are usually taxonomies of these concepts. Likened to the resource description framework (RDF) schema, these classes may be regarded as resources. Resource description framework is a framework for information representation on the web (W3C). The resource triplet, therefore, includes the subject (Analysis technique), the object (ERD/DFD) and the predicate (the IS_A) relationship, as depicted in Figure 5.2.

These relationships, as in the entity relationship diagrams (ERD), represent associations amongst the different concepts (classes) in the ontology. The next component of ontologies consists of the formal axioms. These axioms are used for modelling knowledge, that is, for checking the consistency of knowledge in a knowledge base or for inferring new knowledge in that base (Corcho *et al.*, 2006). Lastly, ontology instances are used to represent elements or individuals in ontology. These are instanced from the ontology classes, for example, an ontology class 'human being' may have men or women as instances or more specifically, John or Jane, depending on the level of granularity.

Structurally, ontologies are used to represent and describe concepts hierarchically in a domain field (Aßmann *et al.*, 2006). They are computationally independent and may be

likened to the computationally independent model (CIM) described in Section 4.3.1. Although they argue that the structure of ontology may be conceived as a collection or group of concepts that are related through associations (relationships), Gonzalez-Perez and Henderson-Sellers (2006) add that the primary purpose of these concepts and, hence, of the ontology is to be able to be used.

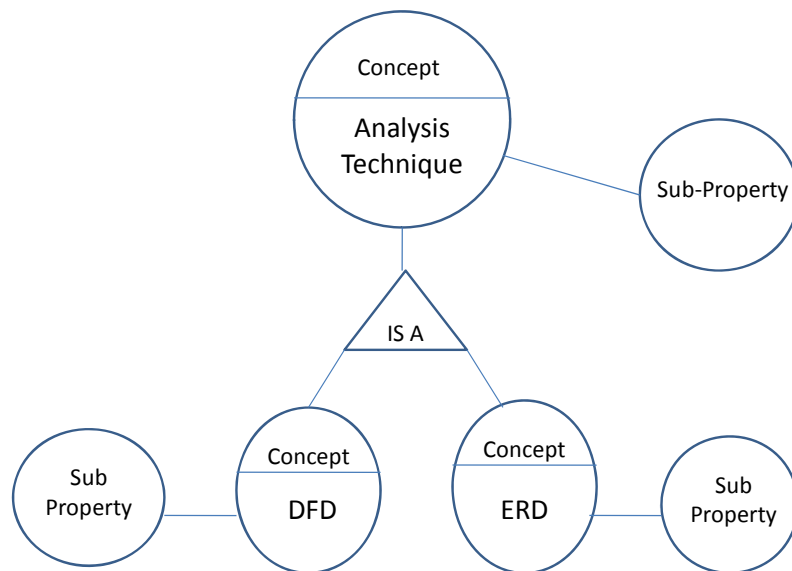


Figure 5.2: Structural Representation of Ontology

Figure 5.2 illustrates three concepts: analysis technique, data flow diagram (DFD) and entity relationship diagram (ERD), all of which are included in the discipline of systems analysis and design. Analysis technique is at a higher level (generalization) than either the DFD or ERD techniques, which are specializations of the analysis technique. The IS A relationship is defined as the association between the three concepts. In addition, each concept is associated with at least one sub-property whose purpose is to fit the concept in a specific domain. It is not surprising that each concept when used in different domains may resultantly have different sub-properties. These sub-properties can also be regarded as concepts and are very important for capturing context.

Each of these concepts is an intensional definition of a set of potential entities and associations between any two or more concepts in a formalization of a group of potential relationships between these concepts. Since the concept has to serve some purpose, the sub-properties of the concepts, being entities that possess the definitions of these concepts, are

also included. These sub-properties complement the semantics in the concepts with pragmatic information. This pragmatic information is subject to change as and when new concepts are created and added to the existing knowledge base of the domain. Any category is thus subject to change, depending on the nature and context of the domain.

5.5 The Concept of Conceptualization

“Formal languages such as logic and computer programs are precise, but what they express so precisely may have no relationship to what the author intended.”

Sowa, 2006.

Authors of prescriptions are always targeting the real world as the item to be mapped. However, the real world has so many states that they cannot all be represented in formalizations such as software products. The gap between what is real and desired and what can be represented can be reduced if real world extensional relations are made to be congruent with the intensional models of software systems. Checkland (1999) argues that the extensional variety of human systems cannot be exhaustively mapped into the conceptual model and that a selection has to be made on what can be modelled and later implemented. This requires analysts to agree on a shared system conceptualization that can be implemented in practice. This conceptualization relies on the ability of developers to create an intensional model of the system. This is all based on what is called a conceptualization. Most of the practical relevance of ontology in information systems results from its definition as a shared conceptualization. This section briefly describes a formal model of a conceptualization.

Using Guarino’s (1998) formal characterization of a conceptualization, a domain space $\langle D, W \rangle$ is defined in which D is the domain and W is a set of maximal states of affairs of D . W can also be referred to as a set of possible worlds or as the requisite variety of a system. A conceptualization, $\langle D, R \rangle$ is thus a structure in which D is still the domain and R is a set of relations, which can be applied in D . Unlike the mathematical definition of conceptualization that emphasizes extensional relations, that is, a description of what an outsider can perceive about the system, Guarino (1998) preferred people to use an intensional account of conceptualization.

Guarino (1998) characterizes an extensional account of a conceptualization as a set in which all elements in a domain are listed explicitly and which can be regarded as a ‘definition by extension’. In terms of possible system states, this definition by extension comprises all

possible system states in the real world. In contrast to this, a specification that defines the property that should be true of each element, the intensional account of a conceptualization, is called definition by intension.

In any system, the intensional relations focus on the meaning of the relationships R as they are applied in D as the domain. These relationships are called conceptual relationships and are defined for relationships in a domain space. In other words, intensional relations are specifications that define the property that each element should possess. Intensional models are, therefore, used to describe what the system should do. Buitelaar *et al.* (2005:1-2) state that the intensional aspects of a system can be used to define and formalize the domain ontology of the system. This assumption is based on the fact that each system possesses application areas or has got intensions. It should be noted that, even if intensional models do not capture all the possible states of a system, they maintain the domain relevance of a system. The following section discusses the different varieties of ontologies that can be used in software development.

5.6 Varieties of Ontology

As discussed in Section 5.5 above, ontologies can be referred to as domain-related, intensional models. There are, however, different ways of classifying these ontologies. These can be classified according to their granularity (Guarino, 1998), functions or form (Fensel, 2004), type of conceptualization structure (Van Heijst *et al.*, 1997) or the nature of existence which Jurisica *et al.* (1999) termed the nature of the actual world issue that needs modelling. In their discussion Ruiz and Hilera (2006) noted the existence of another classification criterion that considers or combines two or more of the abovementioned categories. The bi-dimensional classification proposal as they called it depends on the degree of richness of the internal structure of the ontology and the “subject of the conceptualization” (Ruiz & Hilera, 2006:55). In other words, it is based on the architectural perspective of the ontology, as discussed in Section 5.4.

Since the purpose of ontologies is to capture organizational semantics, Hofferer (2007) uses the semantic expressiveness to group ontologies as having weak or strong semantics. WorldNet is an example of ontology with weak semantics, whereas domain specific ontologies, in which the semantic expressiveness is more than that in WorldNet, may be regarded as having strong semantics. It should, however, be borne in mind that this categorization as *weak* or *strong* is quite relative.

This research is not limited to a single ontology classification criterion, as proposed by these other researchers, but uses these different classifications to arrive at a grouping that suits the requirements of this research, that is, the application of ontologies in the software development field.

As software development is a domain in its own right, domain ontologies have to be regarded first. The research will, therefore, concentrate on domain ontologies, method ontologies and on status ontologies that combine static and dynamic ontologies (as proposed by Jurisica *et al.* (1999), on intentional ontologies, social ontologies and, finally, on process ontologies.

5.6.1 Domain Ontologies

In any study discipline or domain, generic or specific knowledge has to be captured. This usually depends on the developers' requirements. Domain ontologies, as they are known, capture knowledge that is valid for a particular type of domain. Domain ontologies can also be viewed as formal descriptions of the classes of concepts and their interrelationships that describe a specific field of application. As these ontologies have to be reusable, a knowledge base for concepts related to a domain and their relationships should be developed.

During information systems modelling, domain ontologies can capture the domain model and the business model of the system. However, they cannot capture the system requirements since these are prescriptions of the specific system to be developed (Aßmann *et al.*, 2006:266). In effect, domain ontologies play the role of “standardized analysis models” less the specification model. This supports the notion that all ontologies are developed for the sake of capturing information or knowledge that has to be shared within a certain universe of discourse.

If this classification is accepted, neither the internal structure of a knowledge area nor conceptualization need concern the people tasked with sharing knowledge within a specific field. This is because, depending on their requirements, they naturally have to start at the highest level of the structure and work down to the lowest level or vice-versa. Hence the formality and granularity of the domain ontology can be neglected as a classification.

In Section 4.3.1, the analysis model was portrayed as comprising the domain, business and requirements models. This analysis model is a descriptive type of model that conforms to the open-world assumption. During software development the domain ontologies can thus be used to model the domain and business models.

5.6.2 Method Ontologies

Method ontologies capture the knowledge and reasoning needed in the performance of a task. These can be reduced to task or activity ontologies. The basic tenet, therefore, is the ‘what?’ and ‘how?’ of doing a particular task, for example, the requirements-gathering task or the software-design task. Task ontologies, consisting of lexical, conceptual and symbolic aspects are used to specify the objects and their interrelationships that are used in the execution of a certain task (De Oliveira *et al.*, 2006). In short, task ontologies are used to describe a task conceptually.

Although De Oliveira *et al.* (2006) went on to give very sound descriptions of the types of tasks and of their components, it is nevertheless of the utmost importance to derive concepts by which the task ontology is formed from the domain ontology. This process reduces the risk of losing the descriptive nature of the ontology in a domain.

5.6.3 Status Ontologies

In software development we recognize the need to capture and represent the static or the dynamic characteristics of a system. There should, therefore, be both static and dynamic ontologies. These ontologies represent the status of an artefact. Status ontologies, as they are called here, argue for a world containing artefacts that exist and that do not change their form of existence (static), as well as for another class of things that change with time (dynamic). Dynamic ontologies can thus be used to abstract the behavioural characteristics of a system. These characteristics consist of concepts and their interrelationships (Aßmann *et al.*, 2006). As each software development process should allow the modelling of both static and dynamic states of a system, status ontologies are therefore required.

5.6.4 Intentional Ontologies

It must be noted that organizational systems are intentional and their intentions are reflected as objectives (Hirschheim *et al.*, 1995). There must be a way of capturing these intentions in the IS system. This task is given to intentional ontologies. These ontologies are intended to model the softer aspects of living things, of the kinds of things that can have beliefs, desires and intentions. In this category, the human aspects of living things are modelled. Examples of such ontologies are aspect, object, agent and support ontologies, as stated by Ruiz and Hilera (2006). These types of ontologies are also meant to model ascriptions of intentions to actors in a system, as expounded in the discussion of the theory of organized activity (*Section*

4.6.4). Yu (1995) discussed the i^* framework that position organizational actors as having intentional characteristics that is, beliefs, goals, abilities and commitments. This framework emphasizes the importance of actor dependencies in an organizational system. The i^* framework can also be supported with domain and task ontologies as discussed by Mavetera and Kroeze (2010).

5.6.5 Social Ontologies

Social ontologies describe the organizational structure and interdependencies existing among the social actors in these organizations. At a high level of abstraction they include concepts such as the roles and responsibility of the actors, to mention but a few. To enable the social nature of organization systems to be understood, three theories were used in Section 4.6: activity theory (AT), actor network theory (ANT) and the theory of organized activity (TOA) to expand on this. In software development, actor roles, such as those of analyst, developer, tester, to mention but a few, can be used as examples.

5.6.6 Process Ontologies

As Haller and Oren (2006) noted, process modelling activity has been used to describe the intended dynamic behaviour of organizations, as well as to capture the processes and context of these organizations. The organizational context includes data, the roles played in the organization and the resources that are utilized in these processes.

In most system development projects, requirements specify the processes to be used in these systems. These processes, captured as requirements, are usually modelled using the specification model. The specification model is very prescriptive in nature and hence conforms to the closed world assumption. It cannot, therefore, capture the intended system behaviour.

However, process ontologies can capture the three aspects of enterprise knowledge, i.e., context, content and structure. These are reflected as the enterprise knowledge, the domain knowledge and the information knowledge respectively. Enterprise knowledge consists of processes, organizational structure, IT structure, products and customers. Domain knowledge comprises the terms, concepts and their relationships and, lastly, information knowledge consists of the types of documents and document structures that are used in the organization. Process ontologies can capture this triplet as a meta-knowledge model for the organization (Haller & Oren, 2006). Hence, the addition of process ontologies as a specification model

may remove the closed world assumption nature of the specification model. (See Section 4.3.1).

Process ontologies and method ontologies can be regarded as fulfilling the same role in software development. Also worthy of note is the fact that a process repository can be used to capture process models, business rules and checklists, application data and document knowledge and the background knowledge. It should be noted that several ways can be used to classify ontologies. The present classification is not exhaustive and is only included to complement the requirements of this research study.

5.7 Possible Ontology Uses in Software Development

There are several fields in which ontologies can be applied. In this section, we again concentrate on those ontology characteristics that are perceived as being important in software engineering or development. Many software development practices decry the lack of sound communication methods or techniques. The discussion will thus focus on those ontology characteristics that can enhance communication during this process. In addition, modelling, system integration, database integration and software reusability will be discussed. Of all the ontology qualities, communication is regarded as the greatest contribution that ontologies can make to the field of software development.

5.7.1 Ontologies and Communication

In their paper Gonzalez-Perez and Henderson-Sellers (2006) advocate the use of a common vocabulary or conceptual base that can enhance communication amongst stakeholders during software development. They hold that, as an explicit account of a shared conceptualization, ontologies reduce the conceptual and terminological ambiguity of terms among software developers. They also provide a shared cognitive base that can be used in any domain. Through the creation of an ontology-driven knowledge base, the contextual understanding amongst varying software engineering stakeholders can be captured, stored and shared. Aßmann *et al.* (2006) support this, saying that ontologies provide a common (uniform) communication language between the software architect, customer and the domain expert. This increases understanding and interoperability among stakeholders.

The need for an ontology-driven knowledge base becomes critical in environments where software outsourcing is prominent. In these environments, stakeholders in different countries or continents can use the ontology base to share domain knowledge, process knowledge and

task knowledge electronically. An example of this is a case in which an analyst is based in the Americas, a developer in India and the users in Africa.

5.7.2 Ontologies in Modelling

Ontologies have been characterized as models. As models, they can be used in systems modelling. In an attempt to develop an analysis model, ontologies can be used to create domain models, business models and specification models. In such cases they have to be translated to represent a specific group of systems in the domain. Many organizational systems are usually represented using system models. Aßmann *et al.* (2006:254) characterize system models as “models that describe or control a set of systems”. System models should, therefore, comprise both the structural and behavioural models of the system. The role of ontologies in modelling is further supported by the work of Shanks *et al.* (2003), Wand and Weber (1993, 2002) who used ontologies in conceptual modelling and also Guizzardi and Halpin (2008:2) who used them in evaluating “conceptual modelling frameworks”. However, Wand and Weber (2002) urged practitioners to make an empirical determination of the importance of ontologies in software development. In other words, ontologies must be used in an application area to judge their usefulness.

As described on the section on status ontologies (*Section 5.6.3*), the structural model of a system can be captured using static ontologies and the behavioural model can be represented using dynamic ontologies. In short, static ontologies can abstract the structural characteristics of a system. These are “the concepts of a reality and their interrelation, the static semantics of a domain, its context free or context sensitive structure” (Aßmann *et al.*, 2006:254).

Ontologies can be used to provide standardized analysis models. These include the domain-specific models that are developed by different domain experts. The addition of the specification models that translate the system requirements to domain models will yield a complete computationally independent model (CIM) (Aßmann *et al.*, 2006), as described in Section 4.3.1. This process eventually increases the number of behavioural characteristics of the software products.

5.7.3 Ontologies in Distributed Non-homogeneous Database Systems

Ras and Dardzinsky (2004) and Haller and Oren (2006) proposed the use of process or task ontologies in databases. Task ontologies are used to extract rules from different knowledge bases at remote sites. Although global queries can be converted to local queries, the

definitions given to these queries may vary from site to site and interpretations of these may be required. Task ontologies (Ras & Dardzinsky, 2004:55) can be used to define “the computational architecture (structure) of a knowledge system”. The knowledge system performs a task in a domain defined by a set of ontologies called domain ontologies.

Before task ontologies can be used, an intermediate model, as suggested by Maluf and Wiederhold (1997), could be used. An intermediate model describes a database at very high level of abstraction, which is good enough to give a homogenous representation of all the databases in the organization.

Ras and Dardzinsky (2004) noted, however, that the problem with developing systems using task ontologies is that the ontology (domain) repository is kept in a continuous state of semantic inconsistency. Each task may generate a new definition of an already existing concept in the ontology repository. Often the repository may be updated with a somewhat global concept definition while it is, in fact, only a local definition. This problem may be exacerbated by the use of case-based reasoning (CBR) techniques that will force the designs to accept algorithm-generated semantics (Mavetera, 2007; Gomes, 2004). These semantics may not really represent the context of that domain application.

To minimize these inconsistencies, Ras and Dardzinsky (2004) suggested that the semantic changes should be tracked. At each level of information system development, the system should be able to revert to the original definitions given by the users in a specific task-related domain. Mavetera (2007) suggested that a more appropriate way would be to keep the CBR algorithm-generated ontology IS models and designs in a different database from the user-defined ontologies, as discussed in Section 5.8.

5.7.4 Ontologies in Systems Integration

Information systems require social and technical integration to have a fit that benefits the organization. This excerpt on system integration is also described in Mavetera (2004b). Social integration looks at the norms, meanings and group membership of people or their agents. These rules of meaning and membership could be formal or informal. Political structures, although increasingly becoming less effective and less trusted, still play an important role in social integration. Unlike social integration, technical systems integration deals with the technological means of control over the physical and social environment as well as with the skills associated with these means (Lockwood, 1964).

Since both social and technical integration are very crucial for the efficient and effective functioning of an organizational system, the facilitative power of ontologies will be discussed in this section. In system integration, ontologies can be used to allow the smooth transfer of new rules and meanings in organizational systems. On their own, ontologies facilitate the formation of alliances and the establishment of control over the resources that organizations need to achieve their outcomes. These are viewed as translations (Introna, 1997).

Most organizational systems currently in use rely on syntactic standards to work as translation and obligatory passage points. The use of syntactic standards as obligatory passage points, for example, electronic data interchange (EDI), has resulted in large enterprises being inflexible and smaller companies being locked out. In addition, the adoption of a single common standard places limits on the business models that can be supported, thus impacting on the return on investment (Smith, 2000). The use of these standards is, therefore, not strategic if different companies with disparate systems have to be integrated. The standards do not allow for the integration of different organizational political structures, norms and meanings. On the other hand, if ontologies are used correctly, they will be able to address this shortfall.

5.7.5 Ontologies and Software Products Reusability

In line with the philosophy of software product line approach and software kernels, discussed in Section 4.3, ontologies may be used to capture and store the analysis and design models and, in some cases, the implementation model of a system. This model base can be used as a shell for the production of other similar domain related software products. This is also stressed by Mavetera (2007) and is also reflected in Section 5.8 in the discussion of ontology-driven software architecture. This architecture has an ontology knowledge repository in which all the ontology models developed during a software development process are stored and can be reused in other software development projects. There are, of course, several other areas in which ontologies can be – and are being – used but, for the sake of this research, the study will leave consultation of the available literature to interested readers. The next section gives a brief on the role of ontologies in a romantic information system.

5.7.6 The role of ontologies in a Romantic Information System

As discussed in Chapter One, in agent-mediated e-market systems, the use of the schema concept, single state product spaces and negotiation environments have limited the usability of software agents in these e-markets. There is a persistent failure of e-market systems to

capture the meaning, culture, context in the brick and mortar market systems that they are supposed to represent. An ontology-driven e-marketplace will have to capture most of these requirements. For example, the product space can be replaced with ontology of products in the e-market domain. This is captured as domain and business ontologies and kept in a repository. The negotiation protocols can be presented as agent tasks, with rules and resources to use during the process. These can be captured in the system using task, process or method ontologies. For example, tasks such as negotiate discount, compare price and share deal can use resources such as product price, number of participants and delivery period. The tasks can easily be modeled as task or process ontologies that are embedded in the software agent. Intentional ontologies will model the possible life states of the negotiation environment while social ontologies are tasked with capturing the organizational culture and context of the individual organizations as well as the e-market system culture. It must be noted that all seven phases of the agent-mediated e-market framework can be automated using ontology-driven software agents and that will increase the usability of these e-market places. The whole idea is for the software agents in the system to understand each other, use tacit and intuitive knowledge to perform and execute tasks allocated to them by their human principals. These same efforts are being tried for the semantic web.

Another methodological example of ontology use in IS can be found when developing a group of related projects. In this case, ontologies allow “reuse from the modeling to implementation level” of the system development phase (Sharma & Ingle, 2011:147). It is therefore important to have a knowledge reuse development platform that can be structured in the form of the OntoSoft architecture discussed below. In theory and practice as discussed in Sharma and Ingle (2011), there are three phases where ontologies are simultaneously developed and used in software development. The first of these is the *Ontolysis* stage, a requirements engineering phase tasked with the development of domain, business and specification models of the system. This phase ensures the development of domain, business and process or task ontologies that will eventually constitute the analysis model. The second phase is termed *Ontodesign*. This process runs concurrently with the design phase, where the analysis model that is ontology-driven is formalized and all attempts are made to integrate existing ontologies. For quality purposes, ontology evaluation is also exercised. The most important issue at this stage is to pick a formal language that expressly reflects the requirements of the system under development. The third and last phase according to (Sharma & Ingle, 2011) is called *Ontocontation*. This process again runs concurrently with

the implementation phase of the system. The idea is to have an automatic generation of code and case designs as also discussed in Mavetera (2007). The good thing about this is the fact that action or process specific code is ontology driven.

The last example is the use of ontologies in software requirements engineering (Siegemund *et al.*, 2011). Besides addressing the persistent problem of inconsistencies and incompleteness in the specification of requirements, ontologies also capture the business and domain related knowledge (Siegemund *et al.*, 2011, Sharma & Ingle, 2011). This adds also to the fact that, as long as the system is in operation, requirements can never be fully specified as they will continue to evolve. Therefore, ontologies can be used for structuring the concepts, the requirements and relationships captured during the requirements phase. To illustrate this, Siegемund *et al.* (2011) developed and discussed an Ontology-Driven Requirements Engineering (ODRE) tool that, using the power of ontologies, can continuously check the consistency and completeness of the requirements gathered. This will automatically ensure the correctness of the requirements gathered. Of importance in their discussion is the idea that the requirements engineering process is a goal directed activity and this can be carried out efficiently by using business and domain ontologies during the ontolysis phase as discussed by Sharma and Ingle (2011) above. The system specification model is then added as process ontologies to completely develop the ontology-driven analysis model. These application areas are not isolated in their relevance to the need for a methodology that will be discussed in Section 6.10. Sharma and Ingle (2011) pave the way for an ontology-driven software development methodology but theirs falls short of explicitly specifying the ontologies that need to be developed at each phase and what they will address. However, they emphasize the need to capture the softer human aspects of organizational information systems into the software product. Siegемund *et al.* (2011) while supporting Sharma and Ingle (2011) went further to give a pragmatic way of ensuring completeness, hence the fidelity of the system to be developed to the real organizational information system. This can be likened to ensuring quality in the developed system. These examples are not exhaustive but assist in explaining the methodological gap that will be filled by the ontology-driven software development approach when applied in practice.

The following section will discuss an ontology-driven software development architecture that has been conceptualized to become the blue print for a software development environment.

This architecture is necessitated by the fact that most of the modern software development methodologies require an SDE.

5.8 Ontology-Driven Software Development Architecture

Each software development process is carried out in a software development environment (SDE). For completeness, this section will discuss a proposed SDE (Mavetera, 2007) that focuses on the role ontologies can play in software development. The architecture of this software development environment, codenamed the OntoSoft case tool (Mavetera, 2007), that positions ontologies at the centre of a software development process, is illustrated in Figure 5.3. Whitten *et al.* (2004) characterize a case tool as a software package that automates or supports the drawing and analysis of system models. In addition, case tools should provide a facility for translating system models into application programs. The OntoSoft case tool has three major components: the knowledge base repository, the designer engine and the reasoner. The knowledge base repository is discussed first.

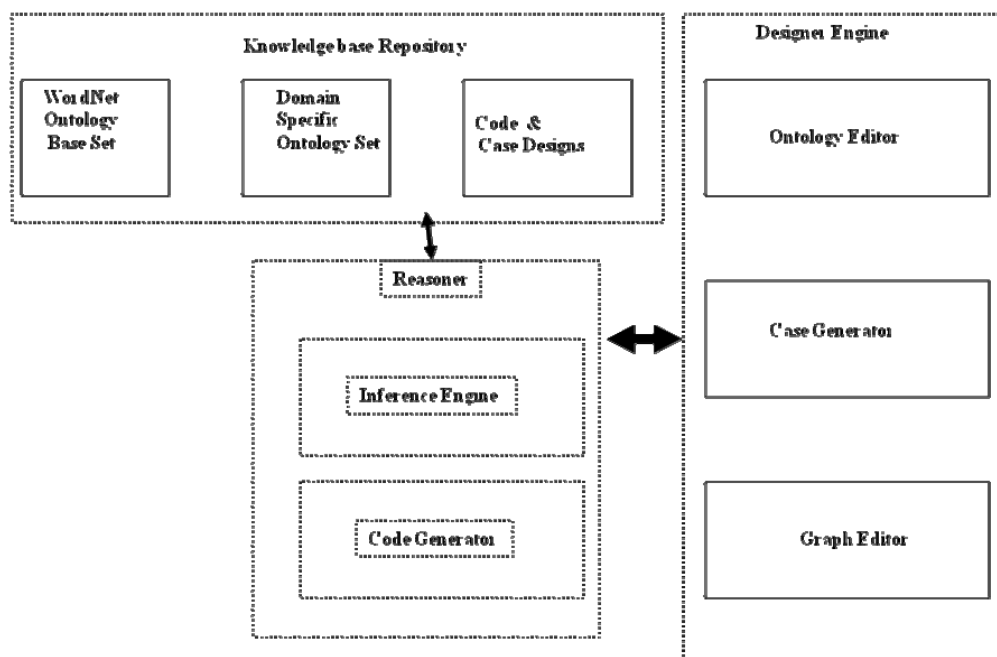


Figure 5.3: OntoSoft Case Tool Architecture (Adapted from Mavetera, 2007)

5.8.1 The Knowledge Base Repository

The knowledge base repository is a facility for storing system models, their detailed descriptions and specifications. These system models are the analysis, design and implementation models explained in Section 4.3.1. The models should be captured as sets of ontologies as well. The tools and facilities for creating system models and documentation should also be stored in this repository. Unlike the Rebuilder case tool discussed by Gomes (2004), the OntoSoft knowledge base repository consists of three parts: the WordNet ontology base, the domain-specific ontology base (not found in Rebuilder or any other case tool yet developed) and a code and the case designs base set (Mavetera, 2007).

The WordNet ontology base is taken and maintained “as is” so as to satisfy the consistency concerns raised in Section 5.7.3. WordNet is a type of terminological ontology (Sowa, n.d.). It is a lexicon and consists of information about “syntax, spelling, pronunciation and usage of words”. The categories in WordNet need not be fully specified by axioms and definitions. It determines the relative positions of concepts with respect to each other, using the sub-type / super-type relationships. In short, it is a natural language knowledge base. In order to maintain linguistic consistency in terms of international grammar and general meanings of terms, this ontology base should not be modified during use.

Unlike the WordNet ontology, the domain-specific ontology set is specific to an application software domain and is allowed to change according to the different conceptualizations and ontological commitments to a certain domain (Guarino, 1998). This is the knowledge base that users and developers can fine-tune to suit their application domains. Finally, the case and code designs base stores new and old designs that are relevant to a specific application domain.

5.8.2 The Designer Engine

This consists of an ontology editor, case generator and a graph editor. These three are used to develop domain-specific ontology case designs ‘*on the run*’. The design engine includes cases that capture the differing designs in the application domain, as well as graphs that are used to map related concepts in a domain through their respective conceptual relations.

The designer engine uses the same principles as Rebuilder in terms of case-indexing, the only difference being the graph editor. Rebuilder uses UML as a case editor. UML, as a case editor, does not link the cases to the meaning of the cases that are stored in the domain

ontology. It is purely syntactic. OntoSoft should use conceptual graphs that are a graphic notation for logic based on existential graphs (Sowa, 2000). The conceptual graphs are augmented with features from linguistics and the semantic networks of artificial intelligence. Conceptual graphs can be used to map to and from natural languages. As a presentation language, they are used for displaying logic in a more human readable form. The conceptual graphs are then linked to the domain-specific ontology to augment the knowledge content of the cases and designs.

5.8.3 The Reasoner

Lastly, the reasoner consists of the inference engine and a software code generator. The inference engine acts as a communication engine between the designer engine and the knowledge base repository. It accepts user queries, retrieves old cases and links new cases to WordNet and domain specific-ontologies and links code to the cases and conceptual graphs. In fact, it is the brains behind the OntoSoft case tool. The reasoner should be able to automatically generate software code that is related and specific to the cases that have been designed.

As such, the code generator automatically develops a code specific to a retrieved or adapted case. This reasoner uses Bayesian Networks (BN) techniques to index cases and case-based reasoning (CBR) principles which are well covered in Gomes (2004). BN uses rules of probability and are incorporated in the reasoner so that it can check for already existing ontologies, case designs, and graph designs. If there are already existing ontologies, case designs and graph designs, the reasoner should prompt the user either to accept or modify them. CBR is, therefore, based on the reuse of experience. Every instance of reasoning is an episode that is registered and stored as a case. These cases are captured as conceptual or existential graphs. It is worth noting that conceptual graphs capture different cases. The individual case captures specific situations that are context related. As each case captures a specific situation and is context related, the syntactic, semantic and pragmatic aspects of the situation are also captured (Mavetera, 2004b).

This research does not cover the concepts on CBR, BN and conceptual graphs in depth since development of ontology artefacts is not the goal. This detail falls outside the scope of the current research. The interested reader is referred to research papers by Gomes (2004) and Sowa (2000 and 2006).

It is stressed that the field of ontologies, their development and use in software development are highly dependent on the field of semiotics as discussed by Stamper (1992), Sowa (2000) and Shanks (1999). There is still some confusion as to what constitutes semiotics. The present study regards semiotics as the study of signs. These signs are used as symbols of representation in a study discipline. As the study of symbols of representation, semiotics focuses on four levels: syntax, semantics, pragmatics and social context. The discipline of semiotics is vast and an in-depth discussion of it will end up defeating the purpose of the present study. The interested reader is urged to consult the relevant literature sources and the researcher limited the scope of his research to the tenets relevant to this study.

5.9 The Field of Semiotics and Ontology

The field of software product development is dependent on the use and representation of the human aspect and the information technology (IT) aspect of systems using signs (Stamper, 1992). These two aspects, as arranged and described by Stamper (1992), portray the IT aspect as having the physical, empiricist and syntactic layers. The human aspect is portrayed as comprising semantics, pragmatics and the social world (context) components in a characterization called the semiotic ladder. These are also referred to as Stamper's (1992) six semiotic levels. However, somewhat thinly, Bjørner (2008) characterizes semiotics as consisting of pragmatics, semantics, and syntax only. This could be because he realized that these three are the levels most commonly used in the field of information systems and are the aspects that are probably the easiest to implement, unlike the organizational context.

According to semiotics, the study of signs (Sowa, 2000; Shanks, 1999) has three branches of representation: the first, second and third, as stipulated by Duns Scotus (1265/66-1308). The signs as symbols should be able to convey knowledge. These branches of semiotics will be discussed in the next section.

5.9.1 Syntactics

Syntax or syntactics refers to the representation and arrangement of signs used in a language. As Bjørner (2008:27) states, it is about representation of specifications, "rules of form, basic forms and their proper compositions". Syntax is regarded by Duns Scotus (1265/66-1308), as '*the first*', branch of representation which is called '*grammatical speculative*'. It is the language that is used to relate signs to each other and to unite material data. Syntactics is also concerned with the forms of symbols rather than with their meaning.

The focus of syntax is on the “how?” of the representation. It defines sets of symbols according to a set of formal rules. The formal definition of syntax allows symbols to be transformed from one form to another. It is important to ensure that the syntax of any language is able to be interpreted in such a way as to assign some specific meaning to the composition. The syntax allows for the compilation and processing of data in computing machines.

In information systems, formal syntax allows computers to merge, share and manipulate ontologies. As such, syntax can be attributed to the correctness and consistency in ontology coding as a form of representation. Of particular importance in syntax is the need for the ontological representation to conform to a particular grammar.

Syntax can also be regarded as signs that are used to map other signs onto a frame of reference. Sowa (2000) used the principle of ‘*concept of representation*’ and conceptual graphs to define and represent syntax respectively.

The use of the concept of representation and of conceptual graphs goes as far as defining relationships between signs. It does not however deal with the issue of relationships among signs, the world, and the agents that observe and act upon the signs (Sowa, 2000). This symbolization uses structured similarity and indices that point at the objects they represent. Regrettably, most of the information systems in use today rely heavily on this symbolization. For example, on the Web, metadata have been used extensively to represent documents and objects using this symbolization. This type of representation does not solve the relationships between the actual objects represented by the meta-level data.

The use of metadata in information systems helps users to find information resources but these metadata are not good enough to process the information so as to find meaning from the resources. Metadata development requires participating information system developers to adhere to some representation standard. This syntactical representation is a superficial agreement. This, on its own, hides the complexities in system relationships and also reduces the interoperability of information systems, as discussed in Section 5.7.4. The next branch of representation is semantics.

5.9.2 Semantics

Unlike syntactics that focuses on the form of representation, semantics deals with the conditions of the truth of representations. As a branch of representation, Duns Scotus

(1265/66-1308) regarded semantics as the second issue to be considered and is defined as logic proper. In organizational systems every sign represents an object that exists in the real world. Semantics is, therefore, used to relate signs to things in the world, that is, to give meaning to symbols that represent the world view. The assignment of meaning to symbols is based on the experience and prior knowledge of the people working with the symbols (Shanks, 1999). It is also the assignment of patterns of signs to related patterns that exist among the things the signs refer to (Sowa, 2000). It is important to note that different actors sharing the same subjective world (the same knowledge and experience) may assign the same meaning to a symbol. This can be compared with the notion of common cognitive base and the definition of ontology given in Section 5.3.

In linguistics, semantics is tasked with the study and knowledge of meaning given to a language. It deals with the meaning given to signs. These signs are, however, usually represented syntactically.

5.9.3 Pragmatics

Pragmatics is usually referred to as the rules of effect. As discussed in Sowa (2000) and also as characterized by Duns Scotus, this can be referred to as '*the third is [...] pure rhetoric*'.

This semiotic level is concerned solely with the way signs and symbols are used. In the use of symbols, there is always a generation of new signs as one works in an application area. Sowa (2000) argues that pragmatics can be regarded as a field that deals with laws that govern the birth and rebirth of new signs. The focus of pragmatics is on the effect of one sign on another sign or the cycle of one thought giving rise to another thought.

As discussed by Bjørner (2008), pragmatics can be viewed as the study and practice of factors that direct our selection of language in social dealings and the effects of our choice on others. This indicates the use of a language in a social context, that is, decisions made to the choice of specific expressions, motives behind specific utterances and expressions. As Bjørner (2008:27) puts it, pragmatics concerns itself with "bridging the explanatory gap between sentence meaning and speaker's meaning". Pragmatics can also be considered as the rules of effect, of influencing others by what we say and how we say it. Sowa (2000) summed it up by suggesting that grammar should be the code for pragmatics.

When applied to information systems environments, pragmatics brings forth the contextual issues that affect a symbol in and around its environment. At this level, the representation

should have a purpose and be useful and usable. Most importantly, the suitability of the representation to the task at hand is brought to the fore.

5.9.4 Social Aspect (Context)

The social aspect is concerned with the context in which a sign or symbol is used. It also deals with the understanding of the meaning that people get when they use symbols. As in semantics, two actors can share the same subjective world, but if they are to assign similar meaning to the symbols, their viewpoints, biases, cultural and political issues have to be the same. A shared understanding of '*the represented*' should be achieved. In other words, the social semiotic level is concerned with the shared understanding in the organizational business world. It is, therefore, important that the cultural and social context of situation be taken into consideration.

This discussion on semiotics is an attempt to find the necessary grounding that can be used to characterize ontologies from a structural linguistic viewpoint. It should be noted that many ontology definitions use a lot of mathematical formalizations, thereby limiting their understandability to practitioners in the hard-design sciences, such as computer science, artificial intelligence, description logics, to name but a few. A softer characterization, based on the field of semiotics is required if ontologies are to be used in methodological fields such as software development approaches. The discussion on the structural aspects of ontologies will be used to find a working definition of ontologies as used in this study.

5.10 Towards a Working Definition of Ontology

The problem with the development of information systems has been the lack of a linguistic model, either informal or formal, that has sufficient translations to allow the exact mapping of the real world view to the system view. In the design of spatial information systems, Molenaar (1998:195) proposed a linguistic model with three components, syntax, semantics and uncertainty, as shown in Figure 5.4. In his discussion Molenaar (1998) says that syntax describes how to link object identifiers to information about the object classes, thematic attributes and geometric data for the formulation of statements about those objects and their mutual relationships.

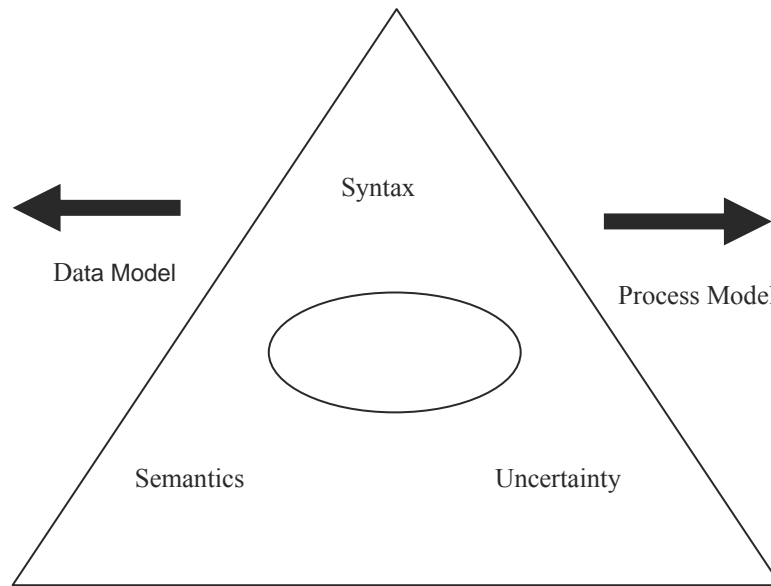


Figure 5.4: A Linguistic Model (Adapted from Molenaar, 1998)

He also portrayed semantics as the link between database representation and a real world situation. In addition, he also noted that the use of data models requires real world phenomena to be described in discrete categories. However this description will always have a certain degree of indeterminacy. Data models, therefore, do not represent reality with absolute precision or certainty. Molenaar (1998) also affirms that all data in information systems refer to objects that are conceptual entities within a certain semantic framework defined within some user's context.

Using context as the focus point, conceptual entities defined by users are, therefore, not very specific. The result is that, in the statement about real world phenomena, terms portrayed in information systems are formulated and understood within a certain application context (Molenaar 1998). This argument shows that any semantic description to be handled by the syntax should also include some uncertainty (fuzziness) inherent in the description of reality as models.

Molenaar (1998) also contends that any linguistic model should be able to handle the correlations of the three components: syntax, semantics and uncertainty, as illustrated in Figure 5.4. Any implementation model for information systems should also be able to incorporate both data and process models within this linguistic model. It is quite uncommon

for Molenaar, as a researcher in the field of natural sciences, not to mention the role of pragmatics in spatial information systems.

In another discussion, Mavetera (2004b) conceptualized a romantic information system development (ISD) model that argued for a linguistic model consisting of syntax, semantics, pragmatics and uncertainty. This linguistic model, without the uncertainty, represented the ontology model. Mavetera (2004b) further argued that, from an ISD paradigm, strategy and methodological perspective, different developers have been addressing the data, process and context needs of systems separately. This has limited the usefulness of the resultant information systems and their software products to a great extent. Furthermore, Mavetera (2004b) did not mention the role of social context in information system in addition to its role on the sound definition of the ontology model presented.

Aßmann *et al.* (2006:256), however, characterize an ontology as a linguistic model, as a “shared, descriptive, structural model, representing reality by a set of concepts, their interrelations, and constraints under the open-world assumption”. As discussed previously, the open world assumption purports that anything that is not explicitly expressed by the ontology is, therefore, (considered to be) unknown. Any aspect of the world view that is known should be captured and documented expressly and explicitly. The intensional conceptualization of ontology models all the features, the known and the unknown, while the extensional conceptualization only captures the known (see section 5.5 above).

This definition argues that the ontology has to be shareable and possess the behavioural characteristics of the domain. Considering the discussions in Section 5.6 (ontology varieties), Section 5.7 (possible ontology uses in software development), Section 5.9 (semiotics) and the work on this section by Mavetera (2004b), Molenaar (1998) and Aßmann *et al.* (2006), the following persistent sticking points are noted about software products, systems and the ontology model:

- Contextual representation requires the capturing of syntactics, semantics, pragmatics and the social context of an organizational system.
- Owing to the existence of informal communication in organizations and some degree of indeterminacy in eliciting implied meanings, there is also some degree of uncertainty or fuzziness in any model that can be used to represent a system in the real world.

- A representation or model whose characteristics are grounded in the relativistic world view that falls in the open-world assumption is required to characterize the real world from the interpretive or neo-humanistic paradigm stance.
- A linguistic model, although being an intensional definition and not an extensional definition of the real world, is required that allows a shared, descriptive, structural and behavioural model of the real world to be captured.

In short, this linguistic model is called ontology and is characterized in softer terms as follows:

Ontology is a linguistic model of the world that comprises of syntactics, semantics, pragmatics as well as the social context of that which is represented. Despite some unavoidable informal indeterminacy in the real world view, it should allow a shared, descriptive, both structural and behavioural modelling and representation of a real world view by a set of concepts, their relationships and constraints under the open world assumption.

This characterization is very important to people who work in the softer fields of IT such as information systems. It is a characterization that allows behavioural and constructivist scientists to develop frameworks that can guide the subsequent development of IT artefacts by the design scientists. Furthermore it also allows them to compare real-life occurrences with what ontologies stand for. For example, in the field of software development, ontologies as a knowledge representation model, communication model, development technique or tool can be contrasted to most of the challenges that developers meet during their daily practices. This definition will be put to use in Chapter 6, in which an ontological approach to software development is formulated.

5.11 Summary

This chapter outlined several definitions that are given to the term ‘ontology’. These range from the fields of philosophy to computer science and information systems. It is argued that, although the philosophical ontology is the basis upon which all the other applications of ontology are grounded in; there are still some differences, depending on the field of application. For example, the purely technical definition of ontology in computer science and artificial intelligence fields may find very limited application in softer fields such as software

and information systems development. A more informal definition, as proposed here, may be more appropriate. This discussion on ontology, besides adding theoretical sensitivity to the researcher, was used as a secondary data input data input during the conceptual mapping process, as will be described in Chapter 6.

CHAPTER SIX

DATA ANALYSIS AND THE ONTOLOGICAL FRAMEWORK

6.0 Introduction

The quest for a development approach that neglects or minimizes the mechanistic and reductionist tendencies of current development methods, but which embraces the humanist characteristics of organizational information systems, was discussed in Chapter 4 and partly in Chapter 5. The issues that currently dominate and dictate software development practices were highlighted in Chapter 4. The gaps existing between the mechanistic and romantic approaches were also highlighted and discussed.

All software development approaches need to be guided by specific paradigms, that is, by some philosophical understanding of how the development problem is viewed and of how knowledge is generated. The reader is referred to Section 2.5 for a brief discussion of the four sociological paradigms suggested by Burrell and Morgan (1979). These paradigms invoke the reader's mind and should be used as a lens when he or she views the nature of a research problem (*See Section 2.9*).

The goal of this research is to find a framework of ontology characteristics and components that can be used in software development. Chapter 5 contained a definition of ontology, its characteristics and how it is used in the development of a myriad of information technology artefacts. Chapter 5 also highlights those ontology attributes that fall into the neo-humanist and interpretive paradigms which can be adapted for use in the development of software products.

This chapter combines the discussions of Chapters 4 and 5 with analyses of interview data from software practitioners in order to derive a framework that can guide the software development process. Starting with the acceptance that knowledge generation using GTM requires the researcher to have an understanding of incidents and categories of those incidents, this chapter uses the issues identified in Chapter 4 together with the researcher's theoretical sensitivity, as a starting point for the generation of categories of incidents from the interview data that were collected for this research. While the problems facing the software development field are quite many, empirical data gathering did not focus on exhaustively collecting these problems from the software practitioners. The major aim of this task was to find confirmatory evidence that supports the need for capturing human issues in software

products. Literature study provides a lot of secondary evidence on problems that are persistently troubling the software development process. However, the empirical study needed to find out if South African software practitioners are also of the view that human issues are important in software development. The purpose of empirical data-gathering was, therefore, to obtain practitioners' views of problems that currently face the field of software development. In addition, suggestions for improving the software development process were also enlisted. A literature study of software development issues provided a secondary source of data that supported the primary data reflected in the data gained from the interviews. This literature study is covered in Chapters 4 and 5 and was a continuous process throughout the research process. This literature study can be referred to as a cross-life activity, as it spanned all the stages and phases of the research period.

It should be noted that the results of any research project are heavily based on the soundness of the methodological process that was followed by the researcher. This consisted of the method chosen, data-gathering techniques, the data samples chosen and the analysis techniques employed. In view of this, this chapter will also document all the procedures followed and the reasons for their use in the research study.

6.1 Profile of Interview Respondents

The data-gathering process targeted people who had experience in the field of software development. The professional experience of the interviewees is, however, varied. The first interviewee had worked as an electrical engineer before becoming a lecturer in computer science, information systems and technology. This interviewee has research interests in software and system development methodologies, particularly in agile approaches and project management. The second interviewee is also an academic and has research interests in data-warehousing and in the philosophy of information systems development methodologies. This interviewee has a vast knowledge of IT. The third interviewee is a seasoned IT professional who previously worked in industry as a developer and is now an academic and a researcher with interests in development of methodologies for both software and systems. These three academically biased respondents were chosen for their ability to provide a wide variety of software development issues, a process that improved the size and number of the codes and categories of incidents generated during open coding.

The fourth interviewee is an IT professional who works in industry as a developer, specializing in military systems, business systems and financial systems. This interviewee

also spent much of his time in IT project management. The fifth interviewee works as a systems analyst in both private and government parastatals. The sixth interviewee is a developer in the financial sector and has a vast amount of knowledge in the field of software development and the software practices currently used in industry. This knowledge is very important in filling those gaps that were not covered by the other respondents. The seventh and last interviewee works as a test analyst and his knowledge, also of software testing, is of the utmost importance, since the problems usually encountered in software development also manifest themselves in the final product. His knowledge of the defects still existing in a software product was used in this research to enable theoretical saturation to be reached. This knowledge could only be collected from a system or software tester.

As indicated above, the first three interviews were with people with a good background in both academics and industry. This decision helped the researcher to elicit both the academic and practical requirements of software development methodologies. Academics tend to have a wider and more general knowledge of software development practices that they gather from reading and teaching students, as well as from academic research. This general knowledge, together with literature in the substantive area of study, is very important during open coding, eliciting categories and the properties of these categories, as well as axial coding.

The second set of interviews was directed at people with a thorough knowledge of their field of expertise, the first respondent having worked in almost all of the different branches of software development, such as analyst, programmer, developer, project manager and system architect, the other two having had extensive experience in the analysis and development of software products. The last interview was with a software tester and, as stated earlier, this immensely improved the theoretical saturation of the findings.

6.1.1 Thematic Areas Considered During Data Gathering

The research interest focused on improving the development process of the software product. As many issues were considered, the questions in the interviews had to be selected to enable most of these areas to be covered. The areas that were discussed during the interviews are listed in Table 6.1. In order to capture as much as possible of the interviewees' knowledge of the software discipline, unstructured, open-ended questions were used. As with all interview processes, some follow-up questions were introduced. The reader is reminded that although the research study focused on developing an ontological approach to software development, ontology knowledge as used in IS is still a very young and developing discipline in South

Africa. As such, questions directly related to ontologies were conveniently avoided. Later, the researcher noted that none of the interviewees had prior knowledge of these ontologies. Instead, the interview themes concentrated on softer organizational issues that are not as yet easy to capture and maintain in software products.

Table 6.1: Interview Themes

Interview Themes	
Paradigm requirements of the software development process	
Software development approach requirements	Capturing of culture
	Capturing of organizational meaning
	Capturing of context
	Capturing of tacit knowledge
Software development method requirements	Communication methods among stakeholders
	Communication among developers
	Communication among developers and users
	Communication of business and domain characteristics from analysis through to implementation phase
	Techniques and tools required and used in software development
Adaptive software development product development	
Software products reusability issues	Requirements reusability, components and knowledge reusability
Software development language requirements issues	
Software success metrics issues	
Software development challenges	

As indicated in Table 6.1, the data-gathering interviews covered several aspects of system development and few dealt with ontologies. The questions were grouped into categories covering the paradigm aspects of software development, communication issues during software development, the software productivity aspects such as quality, schedule and time, the softer aspects of organizational information systems such as culture, context and meaning and, lastly, the role of ontologies in software development. In all, there were twenty six questions whose number could be increased or decreased as dictated by the interviewee's background and profile. For a detailed list of the interview questions, the reader is referred to Appendix A.

6.1.2 Data Sampling and Preparation

The data-gathering process provided the primary source of data, which was also complemented by the addition of secondary data obtained from the literature study discussed in Chapters 4 and 5. The researcher was interested in finding practitioners' conceptions on problems facing the software development process. As indicated earlier, seven respondents were sampled, the idea being to get a broad spectrum of respondents who are software development methodology generalists as well as being specialists in specific phases of the development life cycle. Generalists are described here as respondents who have a vast knowledge and experience of the software development process, both in industry and academia and who have at least worked at each level of the system development life cycle, that is, from initiation to implementation of the project.

With this in mind, the first three interviewees selected consisted of those people whose profiles were described in Section 6.1 above. The data obtained from these three interviewees was perfectly suitable for use for initial open coding (*Section 3.4.5.1*), that is, for deriving the categories of incidents that were later used for the preliminary generation of propositions, as discussed in Section 6.6. Analysis of these first three data samples allowed the researcher to develop preliminary theories that were actually grounded in the software development study area. These theories, presented as propositions, are the direct results of analysis of primary field data.

Having open-coded the first three interview data samples, the researcher collected data from respondents four and five. As these respondents had spent most of their careers in industry and not in academia, their responses had a strong bias towards what is happening in industry. The choice of these respondents strongly supports the aim of the research: to obtain the views of industry practitioners and academic viewpoints as fully and accurately as possible. In addition, the choice was part of theoretical sampling, a process designed to reach theoretical saturation and density in the generated theory. To aid the selective coding process the last two interviewees were carefully chosen, that is they were selectively picked. This time, the idea was to get the views of development practitioners who had vast experience in software development and system testing. The choice of these respondents aimed at establishing the actual use of development methodologies as well as the quality of the software products generated through the use of these methodologies. This part contributed to the theoretical saturation of the propositions.

It should be noted that most of the interviewees were quite conversant with the software development discipline but had very little knowledge of the ontology study area. As stipulated in the research design section, both these study areas are treated as substantive areas. It should be acknowledged that, at this time, the researcher had also studied literature and, being an IT professional himself, the codes and categories of incidents generated were also influenced by the researcher's acquaintance with the literature in the software development and ontology disciplines. This task was done in order to fulfil Klein and Myers's (1999) principle of dialogical reasoning as discussed in section 3.9.5.

With the consent of the interviewees all the interviews were recorded on tape, transcribed and later edited for quality purposes. The interview editing process required the researcher to replay each audio record, at the same time comparing the audio output with the transcribed interviews. This process allowed the researcher to correct any transcription errors such as misspelt words or omissions that could have resulted from poor audio quality or human error. The transcriptions were not edited for grammar and context so as to maintain the fidelity of the transcription to the audio records and the field interviews.

6.2 Components of the Data Analysis Tool Atlas.Ti.

GTM results are heavily dependent on the way the data analysis process is done. This data analysis could be done manually or with the aid of automated software. The presentation style of the results is heavily influenced by the type of data analysis method chosen. In this study, the researcher used an electronic data analysis tool called Atlas.Ti 5.2. Atlas.Ti5.2 is a qualitative data analysis tool that can accept textual, graphical, audio or video materials as input to be interpreted. All the analysis work to be done is contained in a container called a hermeneutic unit (HU). The input documents are referred to as primary documents. In this analysis, seven primary documents, namely P1, P2, P3, P5, P7, P8 and P9, from the field were considered. These corresponded to interviews one to seven respectively. It should be noted that primary documents P4 and P6 form Chapters 4 and 5 of this thesis. These two primary documents (P4 and P6) are as a result of the literature study and were not analysed using Atlas.Ti. They are, therefore, used as secondary data sources.

6.3 GTM Coding Issues

Open-Coding Issues

Relative to GTM data analysis methods, Atlas.Ti captures categories of incidents and properties of these categories as codes and quotations respectively. Quotations are

synonymous with incidents and codes or categories are groups of incidents whose choice was informed by the theoretical sensitivity of the researcher.

The open-coding process uses sentences or lines which make up part of a transcript and, in the case of “*in vivo*” coding, just a word or phrase in the transcript that implicitly or explicitly refers to an incident or category of an incident in the interview. The research relied heavily on the study of literature in order to enhance the theoretical sensitivity of categorizing the incidents.

As discussed in Section 3.4.5.1, GTM recognizes two types of codes: substantive and theoretical codes. The conceptual meanings that are given by generating categories and their properties comprise their substantive codes. These substantive codes are made up of the data patterns that are revealed in the substantive incidents during field data-gathering. The substantive codes (categories) that were developed from this process of open coding and their corresponding incidents are presented in Appendix J on the accompanying CD.

Open coding resulted in revelations from the data that contributed to the initial mapping of M1, that is, M(X) to M(Y) on the double-mapping principle discussed in Section 3.5. Not only did this process start the M1 mapping process, but the researcher was able to generate six propositions from the data obtained from the first three respondents. These propositions were used to refine the two preliminary propositions suggested in Chapter 1.

Axial coding and Constant Comparison Method

Axial coding is the process of searching for relationships amongst coded elements. This process together with constant comparison method can be used to create theoretical codes. In fact, theoretical codes comprise conceptual models of relationships that theoretically relate substantive codes to one another. On that note, codes were further grouped into families of codes. A family of codes is a class of codes that relate to similar occurrences, such as a software development problem or a software development method. This classification of similar codes is part of the axial coding process and the constant comparison method. Table 6.2A lists the families of codes and the number of codes in each family. The last column in this table shows the number of incidents (quotations) in each family of codes. These figures are based on the results of the analysis of the first three interview data samples used for open coding. Also, a full list of theoretical codes and their associated substantive codes that were developed referred to as code families in Atlas Ti.5.2 is presented in Appendix L.

As an example, the code family *Communication technique*, after open coding, is made up of several substantive codes, such as *discussion forums*, *user involvement*, etc. For the communication technique code family, thirteen codes were elicited during the open coding process and are reflected in the *No. of codes* column of Table 6.2A. Then the number of incidents (quotations) that were discovered from the three interview data samples are reflected as *No. of quotations* in this case, twenty three (23) for the communication technique code family.

Table 6.2A: Categories from Open Coding Process

Categories from the Open Coding Process		
Code Family	No. of Codes	No. of Quotations
Adaptive Products Development Technique	1	4
Communication Method	5	11
Communication Technique	13	23
Communication Tool	5	10
Contextual Issues	18	27
Development Approach	10	21
Development Method	11	14
Development Problem	30	52
Development Technique	17	29
Development Tool	4	6
Discussion Tools	1	1
Evolvable Products Development Technique	5	11
Interface Issues	3	8
Knowledge Reuse	2	2
Ontology Requirement	0	0
Semantics	6	8
Syntactics	0	0
Pragmatics	1	1
User Involvement	4	10

According to Mavetera and Kroeze (2009a:16-17), the categories of code families in Table 6.2A can be classified further into communication issues, development issues, semantic issues, quality issues, adaptive and interface issues. This further classification is also a process of axial coding and constant comparison. This classification allowed the researcher to zero in on finding trends and on formulating propositions. Using the network viewer in Atlas.Ti, the relationships between codes and code families were found. The results of this process are shown in Appendix K as a network diagram. The network diagram also allowed the researcher to group the codes and code families further into four thematic areas that are shown in Table 6.2B. It should be noted that ontology requirements have no incidents pertaining to them, supporting the evidence that practitioners had no knowledge of the role that ontology can play in software development.

Table 6.2B: Thematic areas identified from open-coding categories

Identified themes	Code families included
Communication issues	Communication method, technique, tool, discussion tools, user involvement
Development issues	Development approach, development method, development problem, development technique, development tool
Semiotic issues	Syntactics, semantics, pragmatics, contextual issues
Adaptive and interface issues	Adaptive products development technique, evolvable products development technique, knowledge reuse.

These four thematic areas shown in table 6.2B are discussed below.

Communication Issues

From Table 6.2B, communication issues combine the communication requirements and communication techniques used during software development. Effective communication during analysis and design has always been touted as a success factor in software development. The question, therefore, is, *‘What communication methods and techniques allow developers to capture and map the organizational world view requirements holistically to the systems view?’*

The respondents cited several issues which prevail in South Africa. These are listed in Appendix E. These communication issues emphasize the need for software developers to involve the user throughout the project and highlighted the need for tools and techniques such as brainstorming sessions, mind mapping, printable white boards, pair programming, user stories and flyers to be used if the communication problem is to be solved. The respondents also cited the lack of efficient communication methods during software development as a major reason for the failure of software to satisfy users’ needs. It is also important to note that, from the discussions on AT, ANT and TOA (*Section 4.6*), the need for a mediator in the software development process and in the organizational system itself cannot be over-emphasized.

Development Issues

Among the development issues we find a set of development approaches, methods, techniques and tools. Some of these, such as extreme programming and pair programming, are already in use. However the major issues highlighted were the lack of development

approaches and methods that capture the human aspects of a system and later represent them in a software product.

One respondent said that:

“Because software development is not like other IT fields, like traditional engineering and so forth, you find that you can’t come up with blueprints and put them there and say [that] people are going to develop according to this, and they follow that because, basically, things are based on the human brain; it’s more like an art” [P1].

The above statement highlights the need for developers not to design software products using the engineering definitions and techniques used in other industries. Development methods should be interactive and should allow the development of adaptive products. Usually, the choice of a development approach has been based on the schedule requirements of the project and not on the quality and functionality of the product.

Semiotic Issues

Semiotic issues (*Section 5.9*) refer to the signs and symbols that are used for representation and communication in organizations. These issues include syntactic, semantic, pragmatic and contextual issues. The researcher found that current development methods emphasize the mapping of the syntactic software model on implementation platforms, such as programming languages. One respondent said that:

“In our software development processes we tend to focus almost exclusively on the formal part. I’ve certainly not been involved in any information system development project where we’ve tried to create a computer that can sort-of adapt its response to different needs without having to be changed” [P5].

The interviews data analysis results revealed that there is no language that can be used to capture the semantic, pragmatic and contextual requirements of organizations. The findings call for a methodology that captures culture and context in information systems.

Adaptive and Interface Issues

The researcher found that many practitioners wanted to develop adaptive and evolvable software products but that current development methods and platforms did not support these

processes. The user interface was touted as a tool that could be used in communication and also to portray the functionality of the software product. It was, however, noted that the interface could not ensure the development of adaptive and evolvable software products. All these issues, as well as others arising from the interviews, are shown in the appendices.

Developing Theoretical Memos

As the coding process continued, the Atlas.Ti tool was able to capture some theoretical insights as electronic memos. The memos are linked to both the quotations and the codes that they relate to. At the same time, the theoretical revelations or relationships amongst categories (codes) or memos can be shown diagrammatically using the network viewer module in Atlas.Ti. This process is also part of axial coding and constant comparison. Theoretical memos are products of some emergent theoretical insights that struck the researcher during the process of viewing and analysing the data. A full list of theoretical memos developed in Atlas.Ti is presented in Appendix M. In summary, Atlas.Ti helps the researcher by grouping similar codes as code families and similar memos as memo families, at the same time giving a simplified graphical view of the results of the analyses.

6.4 A Practitioner's Classification of Software Development Aspects

The open and axial coding processes allowed the researcher to classify the software development concepts elicited from the three interviews into six categories. These six categories are a result of the researcher's theoretical sensitivity. This sensitivity was enhanced by reading existing literature. Also, the relationships among codes portrayed in Appendix K played a very big role in the development of these categories. This classification is reflected in Table 6.3 as a matrix of software development relationships.

Table 6.3: Software Development Relationships from Practitioners' Perspective

Software Development Relationships from Practitioners' Perspective					
World View	Paradigm	Approach	Method	Technique	Tool
Mechanistic	Realistic	Structured	Incremental	DFDs	
		Structured	RAD	ERDs	Requirements repository
		Object-oriented		Prototyping	Business sponsor
		Data-oriented		DSD	Star schemas
		Process-oriented		Proof of concept	
				Workshops	
				Process chart	
Mechanistic and Romantic	In Transition	Agile		Pair programming	UML
		Object-oriented	XP	Software patterns	Whiteboard
			Prototyping		Flyers
				Software metaphor	Requirements repository
				User involvement	Business sponsor
				Process chart	Star schemas
				Discussion forum	Software metaphor
				Prototyping	Business analyst
				Workshops	System developer
Romantic	Relativistic	Behavioural	Interactive		
		Soft systems	Component	User stories	Requirements repository
		Adaptive	RAD	User involvement	Plain language
		Evolvable			System developer
				Software evolution	
				Software patterns	
				Business analysts	

The concepts highlighted in Table 6.3 are grouped starting from world views through to development tools. Table 6.3 also relates the concepts highlighted to three world views: mechanistic, mechanistic and romantic (M&R) and romantic world view. In Table 6.3 the different paradigms, approaches, techniques and tools are grouped according to the world view to which they belong. As discussed above, this matrix was developed as a process of structuring the data and information elicited from the software practitioners. As may be expected in practice, some of the methods, techniques and tools are employed in more than one development approach, hence their listing in more than one paradigm or world view. This supports one respondent's view that techniques and development tools:

“should not be too much restricted to a particular methodology because, [...] a methodology can actually change, people can come up with different approaches to that methodology, and change the framework and its parameters. So when that happens, the tool itself should actually be adaptable to that.” [P1].

Table 6.3 shows that there are some interesting reflections that appear when one considers the dichotomous relationship between the mechanistic and the romantic world views. It can be seen that the methods and, hence the techniques and tools that are human-centric, have more density in the romantic world view category than in the mechanistic world view. This trend supports the thesis that the capturing of behavioural elements of organizational systems requires methods that will involve users during the development process. The evidence in this table will be used in conjunction with Figure 6.1 in the formulation and refining of the research propositions.

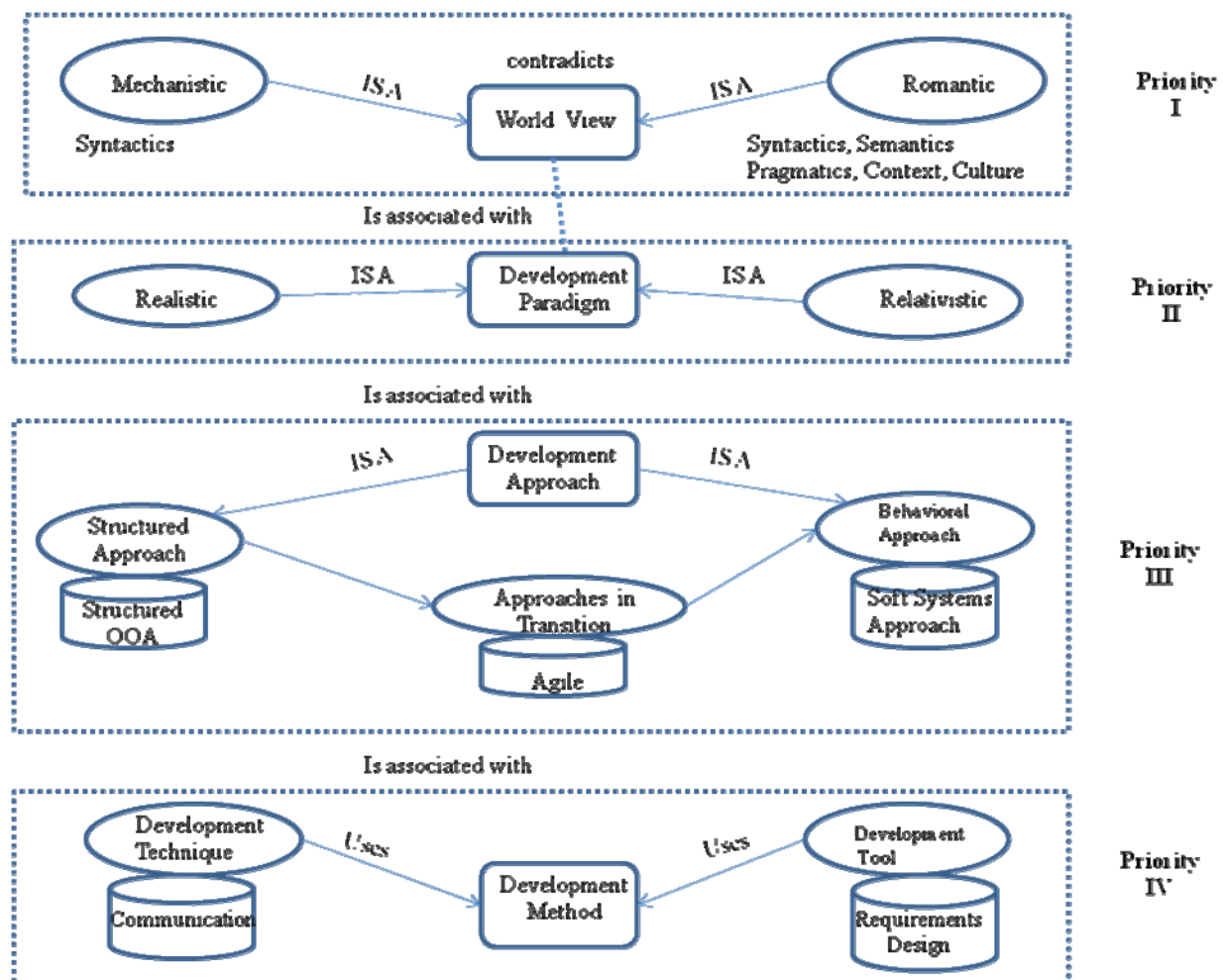


Figure 6.1: Aspects of Consideration in Software Development

6.5 Propositions and Research Questions Revisited

In Chapter 1 two preliminary or generic research propositions were suggested. These guided the researcher on the scope of the investigation to be carried out as well as on the choice of the respondents for the data-gathering interviews.

These initial propositions, as used in this study can be regarded as messages that express opinions that are based on incomplete evidence. This is supported by Leedy and Ormrod (2005:4) who view them as conjectures that should provide direction to the researcher especially on what data to take in order to solve the research problem. As indicated in Section 1.4, these propositions were used as guides, only for directing the scope of the investigation process. After preliminary data analysis, the propositions needed to be thinned out and focused so as to direct the generation of problem-related theory. However, on second thought, the researcher decided to use propositions in place of hypotheses. It must be noted that a proposition is derived as a result of deduction and observing themes and relationships in an empirical situation. The coding of the first three interview data sets has enabled some theoretical trends to emerge and these trends will be used to refine propositions.

Accepting the concurrent use of literature study and data analysis, the propositions were refined using data derived from the open- and axial-coding results, as well as from the literature study discussed in Chapter 4. The information shown in Tables 6.2A and 6.3 was used to develop Figure 6.1, together with the incidents and their categories as elicited from the first three respondents.

In the qualitative data analysis tool used, the relationships portrayed in Figure 6.1 were derived from the memos and network diagrams module of Atlas.Ti. The network diagram is reflected in Appendix K. This process of creating memos allowed theoretical insights that evolved during open coding to be captured for subsequent theory building. The networks module allowed the relationship between codes and the code families to be represented and viewed diagrammatically. This generation of a network of relationships among categories, if used in conjunction with the memos, allows enhancement of theory building.

As depicted in Figure 6.1, four areas of concern were highlighted by the respondents for consideration during the development of software products. Starting from the top of Figure 6.1, these four areas are the world view, the software development paradigm, the software development approach and the software development method. These four areas of concern were concentrated on after it was realized that, despite there being a multitude of issues that combine to define a software development approach, most of these issues could be classified in one of these four primary areas. Each of these four levels is discussed separately.

6.5.1 Refinement of Propositions

Two preliminary propositions were presented in Section 1.4. These are restated here but are now referred to as propositions, as a closer reminder for the reader.

***Proposition A:** The field of software development needs a framework that can be used in the development of romantic software products.*

***Proposition B:** The software development process can be improved by using an ontology-driven approach to software development.*

These propositions will be refined in the light of new revelations from the open-coding results discussed above. As Proposition A does not specifically highlight the issues to be considered during software development, this proposition cannot lead a developer to pick the correct component or steps needed in the development of software products. This proposition should be viewed against the backdrop of questions such as:

- What are romantic software products? and
- What development approach is needed for one to develop these products?

This proposition, if supported, should be able to address the requirements of the first research question in Chapter 1. In addition, Proposition B makes the superficial assumption that ontologies can improve the software development process, without indicating to the reader the characteristics, qualities or components of the ontologies that are needed to bridge the software development gap that currently exists in practice. If this proposition is addressed, it will not only answer the requirements of the second research question, but will provide components or elements of a software development approach needed to develop romantic information products. This will be supporting evidence to address the first research question as well. This section will use the four aspects indicated in Figure 6.1, that is, the world view, the development paradigm, the development approach and method, together with supporting incidents from the field data to refine these two propositions. This exercise thus results in five new propositions being derived.

6.5.1.1 The Software Development World View

The software development process is practiced in an environment that is guided and regulated by the perceptions of the people operating in that environment. These perceptions are found and embedded in a specific world view. In this study, the respondents identified two world views that are different and dichotomous: the mechanistic and romantic world views. In formulating and using a software development approach, one first has to consider the world view in which one is operating. Hence in Figure 6.1, the world views are afforded highest priority (*Priority I*) in the development of a software development approach.

The mechanistic world view conceives of reality as existing and as a given (Struwig & Stead, 2004). Syntactics is the term that holistically describes this mechanistic world view. As supported by the studies discussed in Chapter 4, this world view is heavily dependent on the principles of systematicity, system formation and deconstruction in the greatest totality (Gasche, 1986).

In contrast, the romantic world view posits reality as a social construction (Struwig & Stead, 2004) and believes in a shared reality among actors. Most importantly, in this world view, what may be considered as knowledge depends greatly on the context, organizational politics and culture (*See Chapter 4*). A romantic world view, therefore, consists of semantics, pragmatics, context and culture, as well as of syntactics. This means that the romantic world view encapsulates the human aspects of Stamper's (1992) semiotic ladder.

As can be seen in Table 6.3, a transitional, mechanistic and romantic (M & R) world view can bridge the gap between the mechanistic and romantic world views. This is evidenced in practice, with some approaches using techniques and tools from both world views. As discussed in Chapter 4, both software developers and method engineers (Gonzalez-Perez & Henderson-Sellers, 2006) should not develop or pick a development paradigm or approach before positioning their development problem in the correct world view. The world view generally dictates the development paradigm to be adopted. This discussion on world view will be used in Section 6.5.1.2 to support the presence of proposition (PA).

6.5.1.2 The Software Development Paradigm

As discussed above, every software development paradigm is associated with a world view, whereas an approach is derived from the paradigm. As such, the second priority level (*Priority II*) in Figure 6.1, therefore, depicts the relationship between the world view, the

development paradigm and the software development approach. As in the case of the two world views discussed above, there are also two dichotomous software development paradigms: the realistic and the relativistic paradigms. These were discussed in Section 2.6.2.1 (positivism versus anti-positivism). However, the choice of a software development paradigm has been raised as one of the fundamental issues in the development of software products. The need to choose an appropriate world view and paradigm is captured in the following proposition and in the incidents shown in Appendix C, as derived from the interview data (Respondents P1, P2 and P3).

Proposition (PA): *The field of software development should be placed in a paradigm that facilitates the development of software products that address the socio-technical nature of organizational problems.*

Benson and Standing (2005:3, 7) consider a paradigm as a way of thinking about a specific discipline or “system of working”. They also conceive it as being the basic framework of a discipline (Benson & Standing, 2005). A development paradigm deals with how developers view the nature of the reality that they need to investigate. In other words, it can be considered as the theoretical lens through which reality can be viewed and investigated.

There are currently problems in the capturing of semantics, pragmatics and the social context of a system and in their inclusion in the final software product. As discussed in Section 4.1.1, the purpose of software development is to capture and maintain the patterns of behaviour or the number of organizational system manifestations in order to maintain the original requisite variety in the systems developed. Requisite variety can be likened to the number of possible behavioural states that a system can assume. These behavioural states are most often captured during the analysis phase of software development. Incidents from the interview data transcripts [P2] and [P3] contained in Appendix C are a reflection of the need to consider the paradigm needs of a software-development process.

These incidents support the notion that there are two dichotomous software development paradigms. In addition, they reflect on the merits of both paradigms. Judging from the incident on realistic world view [P2], the positivist stance applies to this paradigm but the relativistic epistemology supports the notion that knowledge is a social creation. The relativistic paradigm accepts the centrality of people in a software development process. It is as a collective that the actual requirements of a software product can be formulated

holistically. In software development, the relativist paradigm urges developers to use approaches that are user- or stakeholder centred.

It can be postulated that the realistic stance has guided the development of mechanistic products rather than of romantic products. The above incidents and discussion lead to the following issues that need to be addressed in a software development process:

- The development approach should assume an interpretive or neo-humanist stance;
- Software development should be regarded as a social construction; and
- A relativist approach to reality should be taken when software products and organizational information systems are being developed.

6.5.1.3 The Software Development Approach

A software development approach is regarded as a framework that guides the development and subsequent use of methodologies that fall within the confines of the philosophical grounding of that approach. This supports the assumption that methodologies using the same approach are most likely to share the same paradigm and world view. Analyses of the data indicate that all three respondents agreed that there was an association between the development approach, the development paradigm adopted and the development method. Development approaches are, however, found between the paradigm and the development method. The study revealed the existence of three classes of development approaches: the structured and behavioural approaches and *approaches in transition*. As indicated in Figure 6.1, approaches in transition lie between the structured and behavioural approaches.

The class of structured approaches, usually referred to as traditional approaches (*see Table 4.2*), include the classical, structured approach and the object-oriented approaches. At the other pole, we find the behavioural approaches, which encompass Checkland (1999)'s widely read, but rarely used, soft systems approach. These behavioural approaches, from which behavioural methodologies are derived, assume a holistic organizational perspective (Benson & Standing, 2005), as well as accepting the social construction nature of organizations, information systems and the software products that implement them. Behavioural approaches, therefore, assume a relativistic paradigm and a romantic world view.

Between these, in the approaches in transition, we find approaches that exhibit both the syntactic characteristics of the traditional structured and the softer humanist elements that are heavily touted in soft systems approaches. These approaches in transition are both mechanistic and romantic. As can be seen in Table 6.3, agile approaches and, hence, agile methodologies make up the bulk of these approaches. There is some debate as to whether or not object-oriented approaches employ the humanistic aspects of the romantic world view. Judging by some of their methods, techniques and tools, as shown in Table 6.3, they can rightfully be included in the class of approaches in transition. It is important to note at this point that the philosophical grounding of ‘approaches in transition’, although it allows the use of techniques that can capture the softer elements of organizations during requirements gathering, does not allow for the inclusion of these same softer aspects into the software product. The techniques do not have facilities for maintaining the domain and business model elements and passing them to the design and implementation models.

Brown *et al.* (2004) support this stance by referring to agile approaches as neo-humanist types of approaches, which stance is partly refuted here as we prefer to call them “partly humanist”. However, it should be noted that, besides the user-centric nature of agile approaches, mostly during the requirements-gathering process, most of the steps in these approaches are inherited from structured techniques. This, therefore, qualifies them to be included in the approaches in transition. The third level in Figure 6.1 cannot be considered without bringing the world view and the development paradigm to the fore. It is allocated priority level III. This initial analysis, also supported by the incidents in Appendix D, allows the researcher to state the following proposition:

Proposition (PB): *The software product development process requires an approach that ensures the capturing of soft elements or behavioural states that are inherent in organizational systems.*

In order to address the requirements of this proposition, several approaches have been proposed, including the agile, object-oriented and even the widely criticized structured approaches, as possible approaches that could be used in varying degrees to capture these softer aspects of organizational systems. However, these evidently do not address the humanistic software development requirements since the software products currently developed are still mechanistic. In Section 4.1.2 it can be seen that the softer aspects of

information systems also include the organizational culture and practice of organizational system users.

Many development approaches, such as agile approaches, employ methods that capture the softer, human behavioural aspects at the requirements gathering stage. However, these are not transferred to the design and implementation phases of the software development process. As discussed in Sections 4.3.1 and 4.3.2, this is because of the lack of an enabling software model that can capture, store and maintain these characteristics down to the lower levels of software development.

The respondents noted that it was most important that the proposed approach employ communication methods that would ensure the effective gathering of requirements that comprise of domain, business and specification requirements. These should also be communicated to the design and implementation models without any loss of the behavioural characteristics of the organization that are captured by the analysis model.

These incidents highlight the need for a development approach that fulfils the following requirements:

- The requirement to capture softer issues of organizations as well as their behavioural characteristics;
- The requirement for a language that captures human behavioural characteristics during development and allows their transfer or sharing among stakeholders;
- The requirement to negate the dominance of traditional approaches and to move towards behavioural approaches;
- The requirement to position this development approach in the relativistic paradigm; and
- The requirement for a developer to act as a tool to reduce the communication gap between users, systems analysts and programmers. This will enhance user requirements-gathering and their faithful transfer to the analysis and implementation models. A developer is regarded here as a person who has both business knowledge and technical skills such as programming.

However, in these incidents, structured and object-oriented approaches are mentioned as the most used development approaches in practice. This, therefore, is evidence to support the reason for software products still showing some mechanistic properties. One can also view it as good evidence to support the requirement for a change in development approaches.

6.5.1.4 The Software Development Method

The development method is shown at the bottom of Figure 6.1 (Priority level IV). This is based on the fact that the approach selected dictates the group of methods that will be used at lower levels in the development of software products. As discussed in Chapter 4, whereas an approach applies to a group of methodologies and a methodology is referred to as a study of methods, the methods themselves are characterized as a way of selecting and using specific techniques and tools to execute a software development project (Bjørner, 2008).

Although not in any particular order or specifying which method uses which techniques, Table 6.3 shows a list of methods and the corresponding techniques and tools that can be used with each method. The reader is again reminded that these methods are only indicative and not exhaustive. Despite the varied number of development methods, the respondents regarded both communication methods, as well as the capturing of human behavioural aspects of organizations during software development, as being the most critical for the success of any given project. On this note, communication methods that can transfer the business model characteristics through all the development phases from analysis to implementation are given more priority over other methodological requirements. Based on analysis of the respondents' replies, there is an over-emphasis on the need for alternative communication methods that can improve the development process. It is thus proposed that:

Proposition (PC-a): *The software development process requires communication methods that ensure that all the stakeholders in the development process understand each other.*

Proposition (PC-b): *The field of software development requires a method or a tool that captures the organizational context during analysis and maintains it through the subsequent development stages of the software development life cycle.*

As discussed in Section 4.1.4, organizational context is captured during domain analysis as a domain model. This context also allows the actors in the organizations to communicate effectively and to understand each other (meaning). The communication method, judging from the varied proposals discussed by the respondents, should require an informal language

that is easily understood by all stakeholders to be used during the analysis, design and implementation stages. A critical look at the communication techniques cited by the respondents, such as discussion forums, brainstorming, proof of concept, including a system or business developer, whiteboards and flyers, requirements repository, users' stories and pair programming, indicates that the most important requirement in such a communication method is to use methods that capture informal characteristics of systems using domain- and business-related concepts. These methods should allow a language community to be built up in the software development environment (SDE) so that all the participants can understand each other. The need for a repository emphasizes the need to capture, store, maintain and share the software requirements within the development team. The incidents shown in Appendix E are used to support these two propositions.

The researcher uses these incidents to support the need for communication methods, techniques and tools that are user-or stakeholder-centric and also use either natural or informal languages for communication during software development.

In summary, the incidents in Appendix E highlight the need for a development method that ensures that it has:

- Communication methods that capture the organizational context and application domain of the system;
- A concept negotiation technique or platform;
- A knowledge-sharing platform for stakeholders; and
- A method that captures the semantics of the system.

In addition, as language limitations constitute the factor that most inhibits the ability of software products to capture informal requirements in systems, the language used should also allow the building of a language community and facilitate knowledge-sharing using concepts. The need for sharing concepts was discussed in section 4.3.1.

Plain language, one that is understood by all the stakeholders, should be used for communication during the requirements-gathering process. It is also important that a business analyst, a person with business orientation, be included to do the analysis. This is like

capturing the domain and business model of the system. The system analysts can then become involved in capturing the system requirements.

The need for a development method discussion revealed several interesting points. These include the use of proof-of-concept as a user requirements verification technique [P1, P2], the use of users' stories for requirements gathering [P1, P3], the use of discussion forums, whiteboards and flyers to capture and share tacit knowledge [P1] and the use of prototypes and interface sketches to communicate the functionality of a system [P1, P2, P3] and user involvement throughout the development process as a method for checking the quality of the software product.

The incidents also clearly indicate that a communication method for capturing semantics is still needed and that some of the current belief that object-orientation captures semantics can be refuted [P1].

6.5.1.5 Adaptive and Evolving Software Development

There is much debate and research in the fields of adaptive and evolvable software product development. This, however, is still a grey area since there are no generally accepted approaches and methodologies that can be used in this endeavour. It is emphasized that adaptive software development is strongly dependent on the approach chosen and used. It is, therefore, proposed that:

Proposition (PD): *The process of developing adaptive and evolvable software products can be achieved by assuming the open world assumption (OWA) principles during the process of system modelling.*

Organizations change their context as and when their environment changes. Since the organizational environment is always in a continuous state of flux and change, the organizational context is always in a continuous state of change, as discussed in Section 4.1.4. To maintain variety superiority over their environment, that is, competitive advantage, organizations have to be able to adapt to the changing requirements of the environment. There has to be a method of capturing the continuous context of organizations in software products. This proposition is further supported by the fact that current design models concentrate on only one single manifestation of the intended system. However, the intended function of a system, as discussed in Section 4.2, is not single, static or discrete, but is always as dynamic as the organizational context. Therefore, there has to be a method of capturing

and modelling the design phase of a system to allow for the capturing of all the intended functions of the proposed system.

In addition, in the translation of the design model to the implementation model, the software product becomes a static implementation of the dynamic organizational system. The paradox of representing a dynamic organizational system as a static implementation model reduces the adaptability and evolvability of software products and of the information systems they represent. This has been – and still – is a problem that continues to face software developers. One of the major objectives of this research is to address this persistent problem.

Despite the confusion in the use of the two terms, adaptive and evolvable, in practice, the two concepts are used interchangeably. However, a software development environment that answers the first three propositions should be able to develop adaptive or evolvable software products. These two propositions are supported by the incidents shown in Appendix F.

The real problem in software development is not caused by the lack of appreciation of the need to develop adaptive software products. The problem is to find an approach or methodology that can be used to develop such products. As reflected in the incidents shown in Appendix F and supported by Section 4.3 (Software Product Lines and Development Practices), the methods currently used do not allow for quick adaptation of the software products to the changing environment. The adaptive and evolvable incidents presented in Appendix F indicate the need for a development approach that encompasses the following attributes:

- The ability to develop software products that can learn and adapt to rapidly changing business environments;
- The ability to develop system upgrades that are fast and easy; and
- The ability to develop software products that can evolve rapidly.

6.5.1.6 The Software Development Environment

To accompany any development approach, a software development environment should be proposed that has the techniques and tools to capture and allow the sharing and reuse of the requirements gathered, of the models and of the designs developed. At the development level, many software development environments have been suggested and are in use. The greatest

requirement for a development environment is the creation of a place in which all the development activities are housed, coordinated and in which the subsequent operational environment of the system is unified. Based on the responses from the interviewees, the following proposition can be made:

Proposition (PE): *The most important requirement for a software development environment is to have a communication protocol that uses a software model (Chapter 4) as a medium for capturing, storing and transferring the characteristics of the analysis model to the design and implementation models without losing sight of the informal domain- and business-related requirements.*

Appendix G contains a list of incidents that support this proposition. It must be noted that most of these issues are not new to the field of software development. However, the fact that they are still being cited as problems warrants their inclusion to support proposition PE. Also the whole idea for the research is to find how these problems can be addressed and hence the need for a solution blue print. From these incidents the following important aspects about the software development environment are noted:

- There is a need for a software development tool or language that is capable of capturing the human aspect of communication. This will capture the informal part of the system in the software product.
- Checking the quality of the software product throughout the development process is essential.
- A development tool is needed that has a requirements repository that captures and stores user requirements during analysis, as an analysis model.
- This repository should be able to be consulted at every stage of system development. Besides improving on requirements communication throughout the project, this also reduces the time to market and the quality of the software products. Requirements are not fully captured because of communication problems as well as time limitations.

- There is a great deal of mistrust between developers and users. This lack of trust leads to poor communication, resulting in poor requirements gathering. An SDE could help to improve this communication requirement [P3].

A software development environment is defined as an absolute, unifying structure of services supporting most (or all) stages of software development and maintenance (Brown *et al.*, 1991). A well designed SDE allows the reuse of system requirements, the sharing and reuse of requirements, the communication of requirements from analysis to design, from design models to implementation models and their reuse. It also creates a language community amongst stakeholders and, in particular, facilitates the communication of user requirements. Brown *et al.* (1991) discuss a SDE that operates at three levels, that is the mechanism, end user services and the process levels, all of which should be integrated into a single environment. Guided by these five propositions, in the next section the researcher revisits the research questions from Chapter 1 in an attempt to redefine and refine them.

6.5.2 Refined Research Questions

This section will discuss the refined research questions to the ones proposed in Chapter 1. As stated earlier, the research questions listed in Chapter 1 were very generic and, according to Glaser and Strauss (1967), initial data analysis is the base upon which new research questions can be formulated.

Main Research Questions

The two preliminary research questions from Chapter 1 were refined and expanded in this section. In these refined research questions and working from the characteristics of mechanistic systems and romantic systems, the focus now shifts from the use of ontology as an artefact in software development to the characteristics found in ontologies. This shift allows the study to migrate from a possible design science bias (Hevner *et al.*, 2004) to a behavioural, constructivist bias as the focus of the study. These research questions will look at the ontology framework of characteristics that are needed to develop romantic software products.

On this note, the first main research question in Chapter 1 is restated and modified here as:

‘What are the components of a software development approach that can be used to develop romantic software products?’

and is further refined to:

- *What are the problems within organizational information systems that inhibit their usability?*
- *What are the problems that are persistent and currently experienced during software development?*
- *What are the requirements of a software development approach that should address these current problems in software development?*

These research questions are directly addressed by the answers given by those respondents working in the software development domain field. As reflected in each of the above propositions, there is a set of requirements that are derived from incidents that support each one of these propositions. With the addition of supporting incidents from interviews with respondents P5, P7, P8 and P9 and of the corresponding axial and selective coding, a complete list of these requirements provides the necessary theoretical ingredients to map M1 that is, M(X) to M(Y), as shown in Figure 6.2. The mapping M1 also includes some of the requirements that were derived from Chapter 4.

AND

‘What is the role of ontology in the development of romantic software products?’

is refined as:

- *What type of software development problems can be addressed using ontologies?*
- *What are the characteristics of ontologies that are important and can be used to represent organizational information systems?*
- *What are the ontology characteristics and ontology varieties that can be used to develop an ontology-driven software development approach?*

This research question is directly addressed by mapping M2 that maps the software development requirements to the different ontology characteristics that are discussed in Chapter 5. Once again, the ontology characteristics were derived from Chapter 5 as a secondary source of data. The next section will discuss the process of GTM theoretical sampling that was done in order to move the propositions towards theoretical saturation.

6.6 GTM Theoretical Sampling and Saturation

Grounded theory requires the preliminary theory generated after initial data coding (analysis) to be densified and saturated by the collection and analysis of additional data samples. This time, the researcher should focus the data-gathering process on those respondents who are most likely to address the needs of the propositions generated. This process is called theoretical sampling. At the same time, the researcher should further analyze or code the data samples that are relevant and support the initial theoretical findings. This process is called selective coding and leads to the theoretical saturation of the generated theory. In this study, the interviews with respondents P5, P7, P8 and P9 provided the data for selective coding, looking for incidents and categories of incidents that supported the initial five propositions discussed above. The supporting incidents were reworded into story lines and those that supported a specific proposition were grouped together using the constant comparison method, as indicated in the following sections.

6.6.1 Propositions PA and PB Story Lines

Propositions PA and PB are supported by the following story lines.

To improve the software development process, a new development approach has to be found. Respondent [P8] preferred a *'a method'* that feeds *'the requirements as direct inputs to a particular modelling or design system, rather than just capturing the [...] what you call the concept of what you want to achieve in the end.'* Current development methods cannot achieve this, *'probably because the languages that we are using restrict us to a specific frame of actions that cannot allow us to go beyond that particular frame.'* [P8] *'There are many times where there are specs but the language that you're using is just not good enough to give you a result for those particular specs...'*

This story line highlights the need for a new development approach and for a language that is more open than that currently used in software development. Also, with reference to Section 6.5.1.2, this story line further raises the need for positioning our frame of thinking into a paradigm that holistically accepts the views, actions and assumptions of the organizational actors. It also includes the requirements for software developers. The field of software development requires a new development approach and a language that does not restrict what developers want to express.

6.6.2 Proposition PB Story Line

Proposition PB is supported by the following story lines.

The practitioners also highlighted many issues that need to be satisfied in a development approach. One of the issues is the requirement that ‘... *the system is built from the perspective of trying to anticipate any - you know, a whole set of ways in which a user would prefer to actually perform a certain function; and then leave it to the user to choose the route that is most comfortable for him or her.*’ [P5]

Respondent [P9] contends that ‘*if we can get the right methodology, which involves users, which knowledge-sharing is enforced*’ then the software development process may be improved. Using mind-mapping in the same way as in agile processes so as to make it interactive, every user will have a card on which the functionality needed can be listed. In addition, a discussion forum should be used to facilitate the sharing of ideas and information.

Things could also be improved if analysts understood the programming platform that will be used to implement the software product. ‘*Many a time analysts do not really know what happens on the programming platform. So, when they give you those specs, they have no idea what’s going to happen in the programming of the application*’ [P8].

The story lines support Proposition PB by advocating an approach that conforms to the open-world assumption that was touted in Section 5.10. Approaches should also allow the development process to use human-centric methods of sharing, communicating data and resources. The need for the analyst and programmer, and possibly for all the stakeholders in the development process, to share the same cognitive base was emphasized. (*See Section 5.3*).

6.6.3 Proposition PC Story Line

Proposition PC is supported by the following story lines. This story line covers both propositions PC-a and PC-b.

Another aspect emphasized in the research was the need for involving the user. User involvement enables the building of a language community during the ‘*practice of eliciting the requirements or systems development.*’[P1]

On mediating when concepts are misunderstood in an organization, respondent [P5] suggested a dedicated room called a *'reality centre and [a process of] modelling things with everybody busily participating in the modelling that takes place... Because, if you are busy developing something like a mind map, you know, you type in the words that you think describe the point that has been made and the people who had made the point immediately see the words you use'*[P5].

On the issue of communication during software development, we use a *'system called Test Director... it captures the actual item on the application that needs to be updated or changed and then a comment as well. And it distributes to whomever the developers or the actual project implementer'* [P8]. *We also have meeting sessions [...] with the users. We have dedicated a session room, which is basically a training room in effect. We've got a PC for each user. ... and we meet with those users in there...'* [P8].

For communicating with stakeholders, *'in Nedbank they were using Microsoft SharePoint, just to distribute ideas. If one has 'an idea on how to approach something,' one will 'write it down on Microsoft SharePoint and everyone in the project will have access to [...]'* [P9] *'In other cases, we'll use your mind maps...'* [P8] Respondent [P8] suggested *'a document of inputs [...] where you actually list everything that's going to be input into the system'* and that the input *'should have a description or explanation to say what it is in actual fact. And that document should actually be circulated to everybody who is concerned in that development process.'* The software development process most often fails because developers are *'not involving the users. Users only get involved 'once system testing has been done...'* [P9].

These storylines support the need for communication methods that allow the building of a language community, the use of natural language as a communication medium and of a platform that allows all stakeholders to know what others are thinking and doing. The basic requirement in the end is a platform where interested stakeholders can come together and share their ideas. This, in brief, concerns communication in the software development phases from project initiation to its implementation.

6.6.4 Proposition PD Story Line

Proposition PD is supported by the following story lines.

Adaptive Software products are viewed as *‘heuristic systems that could learn from the way that a person uses the system and sort-of adapt its response to become more and more in line with how that person uses it.’*[P5] and *‘should leave room for additions or improvements in the future, without constantly having to overhaul the system’* [P8]. The system should *‘actually facilitate for that change or upgrade or anything, without completely having to overhaul the structure or the design of the current system.’*[P8]. It is *‘more of an intuitive system, simply because it readily accepts information and works on it, ... we are in a way bringing an element of intuition into the development process, where you’re basically using that system to do the designs...’*[P8].

With adaptive systems, we are *‘basically trying to build a base of intuition on the application to make it think in a way agreed to say...’* [P8] An adaptive system should be *‘an application that’s open to development and future additions. [Confusion with evolving systems]. Not just in terms of the data itself, but in terms of functionality, in terms of features, in terms of whole lot of other things, you can improve on it in the future.’*[P8]

These story lines highlight the need for a system that learns from its environment as well as from previous experience. The system should also allow the addition of new functionality and for upgrading as and when the context of the operating environment changes. In short, the system should have capabilities for building a base of intuition and should be able to use this to adapt to the ever-changing and running context of the organizational system.

6.6.5 Proposition PE Story Line

Proposition PE is supported by the following story lines.

The most difficult aspect in software development is to store and reuse the gathered requirements. This reuse can ensure quality product development, accelerate the development process as well as provide an easily accessible domain knowledge base. Respondent [P2] said: *‘I don’t think the end-user will benefit as much as the developer from doing it [having a requirements repository] because you have one*

version of the truth of the requirement, you have one place - central repository - where you can find the requirements, you can set up checklists to see if you have fulfilled the requirements. The requirements are nearby but then the system should force the developers or the analysts to fill it in'. And the 'client should be there throughout and that is basically for quality check purposes, to ensure that the kind of requirements that the user has are actually met' [P1].

'...to ensure that requirements can be re-used, you need to have a sufficiently organized approach to your software engineering so that you can have access to the designs and specifications and architecture of systems that have previously been built' [P5].

'I would document all my processes or my solutions, I'll put them in repository and then from there I can basically work on those repositories and documentation' [P8]. It is important to 'create objects that allow you to reuse them, regardless of the different environment, or organization, or application you decide to use them in.' [P8].

From the software testing perspective, the user ensures quality gates in the development process [P9]. *'So, if you interact with the users you'll get less support calls. So interact with the user from the first stage until the implementation stage.'* [P7]

Another reason for involving users at all stages is *'so that we don't spend a lot of time on things that's not important.'* [P2] *'In other words, a collaborative approach where the client is really a part of the team.'* [P5] *'So the user involvement is always the key in terms of software development,'* [P9], because the user (client) is the one who checks the conformance of the system with business requirements and functionality.

The story lines for Proposition PE highlighted issues such as advocating a software development environment that could, *inter alia*, store and reuse requirements gathered during system-analysis process and act as a repository that maintains the users' understanding of the requirements. Such an environment would improve the way in which developers churn out their software products. The repository could be used to obviate the need for a client always to be available during the development process. Respondent P5 laments the non-involvement

of clients in the development process and states that this is one reason why software products fail. To summarize, these story lines argue that there would be a significant reduction in costs and a corresponding increase in software quality if a repository were included in a software development environment. In the end, the repository provides a storage facility for all the system documentation.

These story lines, including the discussion in Section 6.5.1, have been synthesized to come up with a full list of software development process requirements which, if addressed, would have a very good chance of improving the software development process. The full list of these requirements is listed in Table 6.4.

It should be remembered that story lines 6.6.1 through to 6.6.5, together with the additional requirements generated from these, are the result of the processes of selective sampling and coding in an attempt to reach theoretical saturation. By and large, the findings from these story lines are similar to the requirements derived from the open-coding process. The research is, therefore, considered to be theoretically saturated and a final frame of software development requirements can now be compiled. Before the components of the final requirements framework are listed, it is important that the researcher revisit and modify the GTM double-mapping principle discussed in Section 3.5.

6.7 The Double-Mapping Principle Revisited

As discussed in Section 3.5, the double-mapping principle in this study was necessitated by the presence of two substantive areas of investigation. In brief, the researcher needs to map occurrences in the field data to some of the concepts in the first substantive area, the software development study area. But this first mapping, M1, does not fully address the requirements of the study, that is to come up with 'An ontology-driven software development approach', where, *inter alia*, an ontology-driven framework has to be developed. This, therefore, necessitates a second mapping, M2.

Figure 6.2 is an expanded version of Figure 3.1. Using Figure 6.2, the first mapping M1= M(X) to M(Y) remains the same as that discussed in Section 3.5. This M(Y) is a complete list of software development requirements elicited from the data analysis of all seven interview data samples, as well as from the findings from Chapter 4. In a study consisting of a single substantive area, the relationship between the categories in M(Y) could be used singly to

generate the substantive theory. The output of this mapping is presented in Table 6.4 as a requirements framework for a software development approach.

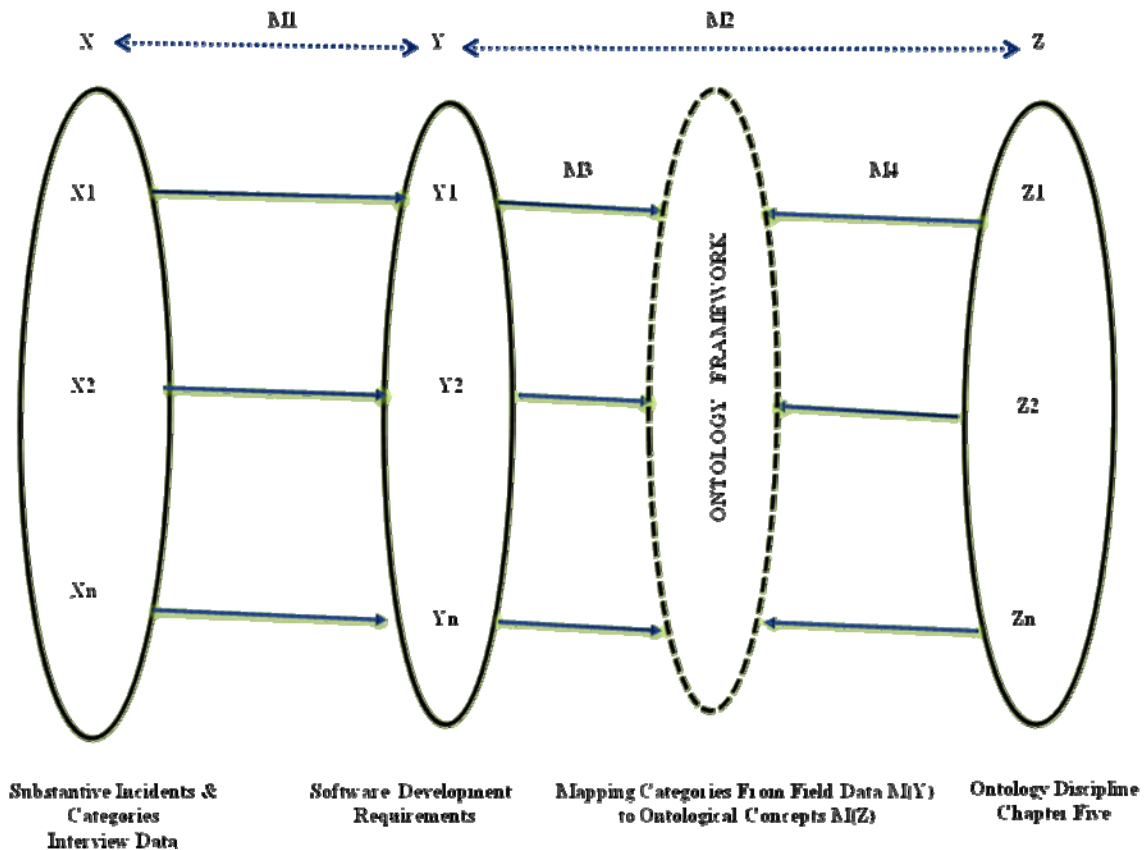


Figure 6.2: Expanded Double Mapping Principle

6.8 A Requirements Framework for a Software Development Approach-The What Part.

To prevent any possible loss of continuity and clarity, the requirements of the software development approach are grouped in Table 6.4 under the headings proposition PA through to proposition PE, depicting their presentation in Section 6.5 in which they were originally classified. The reader must note that this requirements framework is not an exhaustive list of all the issues that are needed to solve all software development problems. It is only an indication of problems that are still gray when considering the capturing of softer human aspects of organizational systems. It must be noted that, in addition to the issues derived from the interview data, GTM also allows data from literature study to be added as supplementary sources when building a theory. In Table 6.4, therefore, some of the requirements are a result of literature study presented in Chapter 4. Hence, Chapter 4 is regarded as a primary document P4 for the purpose of this analysis.

Table 6.4: Requirements Framework for a Software Development Approach

Requirements Framework for a Software Development Approach
<p>Proposition (PA): <i>The field of software development should be placed in a paradigm that facilitates the development of software products that address the socio-technical nature of organizational problems.</i></p>
<p>Requirements:</p> <ul style="list-style-type: none"> • The development approach should assume a neo-humanist stance. • Software development should be regarded as a social construction. • A relativist approach to reality should be taken when software products are being developed. <p>These are supported with evidence from section 6.6.1</p>
<p>Proposition (PB): <i>The software product development process requires an approach that ensures the capturing of soft elements or behavioural states that are inherent in organizational systems.</i></p>
<p>Requirements:</p> <ul style="list-style-type: none"> • The software development process should capture the softer issues of organizations together with their organizational behaviour. • The software development process requires a language that captures human behavioural characteristics during development and that allows for their transfer or sharing among stakeholders. • The software development process should negate the dominance of traditional approaches and move towards behavioural type of development approaches. • The software development process should have a developer as a tool that reduces the communication gap between systems analysts and users. This would enhance user requirements-gathering and their faithful transfer to the analysis model. • The software development process should capture and maintain the patterns of behaviour of organizational systems (<i>supported by Section 4.1.1</i>). <p>This is further supported with evidence from section 6.6.2.</p>
<p>Proposition (PC-a): <i>The software development process requires communication methods that ensure that all the stakeholders in the development process understand each other.</i></p>
<p>Proposition (PC-b): <i>The field of software development requires a method or tool that captures the organizational context during analysis and maintains it through the subsequent development stages and the development life cycle of the system.</i></p>
<p>Requirements:</p> <ul style="list-style-type: none"> • The software development process requires a method that captures the ever-running organizational context and application domain of the system. • A process or method should be in place for checking the quality of the software product throughout the development process. • The software development process requires a concept negotiation technique (<i>supported by Section 4.1.3</i>).

- The software development process requires a knowledge-sharing platform for all stakeholders (*supported by Section 4.1.3*).
- A method that captures the semantics of the system is required.
- There should be a method that allows developers to capture the culture and practice of organizational system users and to maintain these in the software product (*supported by Section 4.1.2*).
- There should be a method of capturing and modelling the design phase elements of a system to allow for the capture of all the intended functions of the proposed system (*supported by Section 4.2*).

Proposition (PD): *The process of developing adaptive and evolvable software products can be achieved by assuming the open world assumption (OWA) principles during the process of system modelling.*

Requirements:

- There should, therefore, be a method for dynamically representing an organizational system as a dynamic piece of software.
- The software development process should develop software products that learn and adapt to rapidly changing business environments.
- The software development process should allow for system upgrades that are both rapid and easy.
- The software development process should enable evolvable software products to be developed.
- The software development process should make provision for the capture, storage and maintenance of the organizational business model throughout the development stages (*supported by Section 4.1.3*).

This is supported with evidence from section 6.5.1.5

Proposition (PE): *The most important requirement for a software development environment is to have a communication protocol that uses a software model as a medium for capturing and transferring the analysis model characteristics to the design and implementation models, without losing the informal domain- and business-related requirements.*

Requirements:

- There is a need for a software development tool or language that is capable of capturing the human aspect of communication. This would capture the informal part of the system in the software product.
- The software development tool should also allow the building up of a language community and facilitate knowledge sharing using concepts.
- There is a need for a development tool that has a requirements repository that captures and stores user requirements during analysis, as an analysis model.
- The requirements repository should be able to be consulted at every stage of the system development process. In addition to improving requirements communication throughout the project, this would also decrease the time to market and improve the quality of the software products.
- Requirements are not fully captured because of communication problems and time limitations. Therefore, the requirements repository/tool should improve the way requirements are captured in terms of alleviating communication problems and reducing time.
- Plain context-related, domain language that is understood by all



stakeholders should be used during communication when doing requirements gathering.

- The repository should allow the reuse of previous requirements.
- In order to reduce the time taken when case tools are used during software development, there is a need for a software development environment (SDE). A methodology dictates the way a software engineering environment is subsequently used. If this is lacking, introduction of the software engineering environment on its own may cause problems of fit. Since user requirements-gathering takes up 80% of the development time, it is important to have a development tool that speeds up the process. Without this, there is a risk of developers rushing the requirements-gathering process and implementing an incorrect system.
- As there is a great deal of mistrust between developers and users, leading to poor communication and resulting in poor requirements-gathering, a tool is required that captures or negotiates between users and developers (*supported by Section 4.2*).
- A way of reusing domain knowledge in software development is required, as discussed by De Oliveira (2006).

Other software development requirements

- A tool is required that is not restricted to a single methodology [Respondent P1] but can be adapted to suit other methodologies.
- The method of use of development tools should not be too prescriptive.
- Developers should be afforded enough time to draw up the architecture requirements of the software product as a quality assurance measure.
- A tool is required that is open enough to allow users to use their expertise and skills [Respondent P1].
- There is a requirement for a new programming language that can be used to implement the human aspect of organizational systems [P8].

The following problems, that are still grey areas, were identified from a literature study of the software development substantive area:

- Understanding of users' practice in an organization requires time (*Section 4.1.2*).
- There is lack of temporal alignment between implemented, situated practice and current organizational practice (*Section 4.1.2*).
- The capturing and sharing of concepts in organizations is still a problem (*Section 4.1.3*).
- Understanding of the business environment before system requirements-gathering starts is a problem and takes a lot of time.
- The software development process lacks a method for identifying, capturing and communicating context in the software product (*Section 4.1*).
- The "where, when and how" to capture, communicate and store context in the development process is a problem and is still a grey area.
- Software developers lack an appreciation of the domain knowledge (*Section 4.2.1*).
- The sharing and reuse of knowledge is very limited (*Section 4.2.1*).
- There is a failure of the analysis phase to create a true reflection of the organizational system environment.

It should be noted that a myriad of methods, techniques and tools can be developed in an attempt to satisfy the requirements presented in Table 6.4. A study of literature revealed that no single method can address all these requirements. At the same time, these requirements are not new to practitioners and researchers in the field of software development. In eliciting these requirements from practitioners and researchers in South Africa, as well as supporting this with evidence from literature study (Chapter 4) it is most important to highlight their persistence and to impress upon the reader that these problems are still current. This should allow methodology engineers to have some moment of reflection and ask questions such as ‘So what?’ ‘What then?’, ‘*Quo Vadis* methodologically?’ ‘To what is the research of software development methodologies leading and what are we as methodology engineers doing in our attempts to address these issues?’

Researchers really understand where they are coming from, judging from the several attempts they have made to try and address these requirements, but, needless to say, the software development process is still in a ‘crisis’. The requirements framework of Table 6.4 has also integrated the discussions of Chapter 4, the literature study on software development issues, with the empirical results obtained from interviews with software practitioners. It is the “what?” of software development requirements. This integration is also an attempt to indicate to the reader that, in practice, the problems that keep coming up during software development are not completely new. One is, therefore, encouraged not to look for novel problems in software development, but for novel methods of solving these problems. It should also be noted that, in the requirements framework, cross-references have been indicated that show the existence of these problems as documented in previous studies and discussed in Chapter 4. In Table 6.4, any requirement that is not cross-referenced was directly derived from the interview data.

6.9 The Ontology-Driven Software Development Framework–The How Part.

This section discusses the ontology-driven software development framework that will later be used to develop an ontology-driven software development approach. A framework is conceived by Sarantakos (1997) as an artefact that emerges from experience and is revised and corrected through several research studies. Most importantly, it does not act as a “blinder or strait-jacket” but should be directed and fine-tuned to serve the needs of the research study

discipline. There are several key elements that can be bundled into a framework. These are listed below:

- A framework should explain the key factors or variables of a study problem. These define the dimensions or spectrum of the study problem.
- A framework should provide a description of the presumed associations or links between these factors and variables.
- A framework should list the nature of the problem (the what?) and the theoretical constructs to be used in the study.
- A framework should list the expected results, outcomes or findings.
- A framework should be a trajectory on which the problem should be solved and, lastly,
- A framework should coordinate the activities of the project or study team that intend to solve the problem.

A framework, therefore, can be described as a conceptual structure that can be used to guide solutions to problems. This ontological framework, in particular, proposes ontology components that should be incorporated into software development practices. According to Gregor (2006:623), this is a Type I theory, which “gives rise to a description of categories of interest”.

In Section 6.8 above, the software development requirements need to be transformed using M2 of Figure 6.2. This is the second mapping, $M2=M3+M4$, which is required to map the software development categories $M(Y)$ to some set of ontology concepts to which they relate, $M(Z)$, that is, the transformations M3 and M4 illustrated in Figure 6.2. This second mapping became necessary because of the existence of the second substantive area and the research requirements of finding a framework of ontology elements that could be used to improve the software development process. As discussed earlier, these ontology elements do not directly have the same meanings as the concepts in use in current software development practices.

It should be noted that the interview research questions depicted in Appendix A could not explicitly refer to the ontology characteristics required in software development because the field of ontologies as used in information systems is quite new in South Africa. Any attempt

to refer to the ontology artefact as used in IS would have been counterproductive as very few, if any, developers currently practicing in South Africa have an idea of this discipline. This would also have confused the interviewees to the extent that they could question the purpose of the research interest. In order not to lose the interviewees, the questions concentrated on the software development practices and a mapping, M2, was used to transform the software development requirements to the ontology attributes needed in software development. This mapping can also be used to judge the theoretical sensitivity of the researcher.

In this mapping, it is emphasized that the mapping M3 is a reflection of M(Y) and that M4 is a reflection of M(Z), that is, of the ontology characteristics as coded by [P6] (*Chapter 5*). The reader is again reminded that Chapter 5 acted as a primary document in which case, the information reflected in Chapter 5 provides a secondary source of data used in this study. It is important to note that in terms of GTM dicta, this is also a process of theoretical coding. Mapping M2, therefore, matches a set of ontology elements M(Z) through M4 to a set of software development requirements M(Y) to M3. The result of this mapping is reflected in Figure 6.2 as the ontology-driven framework for use in software development. The results of this mapping M2 are shown in Table 6.5 below. In short, Table 6.5 is equivalent to mapping M2.

Table 6.5: Ontology-Driven Software Development Framework

Ontology-Driven Software Development Framework		
Issue	Software development requirements	Ontology aspects
1	Capture domain and business model.	Use domain ontology (<i>Section 5.6.1</i>).
	Maintain business model and domain model characteristics from analysis to implementation.	Use of domain ontologies as software model (<i>Section 5.6.1</i>).
2	Capture possible life states of a system.	Ontology is an intensional model of the system (<i>Section 5.4.1</i>).
3	Capturing system requirements- requirements specify the processes that run in organizational systems (specification model).	Use process/method/task/activity ontologies (<i>Sections 5.6.2 and 5.6.6</i>).
	Capture specification model.	Method and process ontologies capture the knowledge and reasoning needed in performing a task (<i>Section 5.6.2</i>). Concepts used in task/method/process ontologies should be derived from the domain field so as to reduce the risk of losing the descriptive nature of the method ontology in the domain.
4	Capture the requirement-specification model in a domain-related language.	Use method or process ontologies (<i>Sections 5.6.2 and 5.6.6</i>).
5	Avoid errors during software development. Improve software quality.	Domain ontologies carry the domain knowledge into the software product itself (<i>Section 5.2</i>) and into the maintenance stage.
6	Capture the descriptive analysis model – the analysis model is a descriptive type of a model that conforms to the open-world assumption (<i>Section 5.6.2</i>).	Use domain, process or method ontologies (<i>Sections 5.6.1, 5.6.2 and 5.6.6</i>).
7	Capture the behavioural attributes of systems, i.e. static and dynamic attributes. Software development should allow for modelling of both the static and dynamic states of a system.	Status ontologies can capture the static (change in form of existence) and dynamic (time dependent) aspects of the organization (<i>Section 5.6.3</i>).
8	Capture behavioural aspects of the system.	Use intentional ontologies –they model ascriptions of intentions to actors in a system (<i>Sections 4.6.4 and 5.6.4</i> .)

9	Capture organizational culture and context.	Use social ontologies – these model organizational structure and their interdependencies (<i>Section 5.6.5</i>), such as roles and responsibilities.
10	Reuse of ontology-driven analysis model.	Use ontology-driven repository (<i>Section 5.8</i>).
11	Reuse of software requirements.	Use ontology-driven repository (<i>Section 5.7.5</i>).
12	Ensure quality, reduce cost and keep to scheduled times.	Use ontology-driven software development environment (<i>Section 5.8</i>).
<p>The software development process requires an ontology-driven analysis model that is made up of domain, process or method ontologies to capture the domain, business and specification models of the system to be represented as a software model. This should be stored and maintained in an ontology-driven repository. For addressing the requirements of the software development metrics, that is, in order to improve schedule times and quality and reduce development costs, an ontology-driven software development environment should be used, as proposed in this study.</p>		

In support of the points listed in the framework of Table 6.5, Ding and Foo (2002:124) also noted that ontologies can improve “information consistency and reusability, systems interoperability and knowledge sharing”. They capture and describe the semantics of a specific domain in a manner that is both machine processable and human understandable. At the same time, this is not the first time that ontologies have been tasked with roles of this nature. Falbo *et al.* (2002) developed an ontology domain engineering approach that could integrate ontologies and object-oriented technology in the development and reuse of software products. In their approach, object models at the analysis, design or even at the implementation stages could be derived from an ontology knowledge base, that is, almost fulfilling the same role as the one proposed by Mavetera (2007) and also discussed in this thesis. Unlike the framework presented in Table 6.5, their approach lacked the step-by-step process of doing such an endeavour. They actually decried the deficiency of approaches that can be used to add ontologies in a more conventional software development process (Falbo *et al.*, 2002), describing this as a major limitation for the use of ontologies in software engineering.

Ontologies also reduce the pitfalls usually found in system connectivity efforts that rely on the physical and syntactic layers of the semiotic layer, as discussed in Chapter 5 (Ushold and Gruninger, 2004). When it comes to software maintenance and reuse, developers have only one locus of information that they need to update if the application area evolves, that is, the ontology knowledge base. The domain ontology that needs to be developed during software

development is made up of concepts and of the relationships between these concepts. It is an exhaustive catalogue of the concepts that are found in the domain field. This catalogue is what the software to be developed needs to capture. It also acts as an external source of documentation for developers. From the discussion in Chapter 5, there are several practical applications in which ontologies can be used during software development. This framework is, therefore, not exhaustive. Developers should note that there are many areas in which an ontology-driven framework for software development such as the one presented here, can be applied. It should be noted that the framework presented in Table 6.5 lacks some application context in terms of a formalized way of using it. A probable approach that could be used to operationalize this framework is discussed in Section 6.10.

6.10 An Ontology-Driven Software Development Approach-The How Should Part.

In Section 4.7 an approach is defined as ‘*a set of goals, guiding principles, fundamental concepts and principles for the system development process that drive interpretations and actions in system development*’. These approaches combine methods that share goals and principles for the software development process. A methodology can be derived from the approach, using the underlying principles and the fundamental concepts of the approach.

Since the term ‘*principle*’ is used quite often in this document, it is important that its definition in Section 4.7 is repeated here. In Section 4.7 a principle has been described as ‘*an accepted or professed rule of action or conduct...*’. An extra task in this research study is to combine the meaning of an approach with that of ontology in order to arrive at an ontological approach to software development.

Several definitions given of software development ontologies were discussed in Chapter 5. These were critiqued and the researcher finally proposed a working definition of ontology for this study. This definition in Section 5.10 is restated as follows:

Ontology is a linguistic model of the world that comprises syntactics, semantics and pragmatics, as well as the social context of that which is represented. Despite some unavoidable informal indeterminacy in the real world view, it should allow a shared, descriptive, both structural and behavioural modelling and representation of real world view by a set of concepts, their relationships and constraints under the open-world assumption.

This definition of ontology is the foundation upon which ontologies can be used in software development. To qualify as an ontology-driven approach to software development, the majority of the methods, techniques and tools that are used, as well as the tasks that are performed during a software-development process, should ensure conformance to this definition. As motivated in Chapter 5, Section 6.9 and in Table 6.5, ontologies can address many of the software development problems that lead to mechanistic and non-adaptive software product development. The research will, therefore, use both this definition and the framework of Section 6.9 to develop an ontological approach to software development. The components of the approach according to the goals, guiding principles, fundamental concepts, principles and actions in software development that drive the interpretations in this approach are listed in Table 6.6 below.

Table 6.6: An Ontology-Driven Approach to Software Development

An Ontology-Driven Approach to Software Development	
Goals	<ul style="list-style-type: none"> • To capture and maintain behavioural aspects of organizational systems in software products. • To develop adaptive and evolvable software products. • Once software requirements have been captured, to store, maintain and reuse them without having to re-enter them in the software development platform.
Guiding Principles	<ul style="list-style-type: none"> • Allow mapping of organizational environment to the analysis model. • Allow analysis model characteristics to be translated to design model without loss of domain and business characteristics. • Allow analysis and design characteristics to be maintained in the implementation model. • Allow the reuse and sharing of analysis, design, and implementation models during the software development process. • Allow automatic generation of design cases from the analysis model.
Fundamental Concepts	Approach, methodology, method, technique, tools, project management, software development, software engineering environment, ontology, domain ontology, method ontology, process ontology, analysis model, design model, specification model, implementation model, business requirements, domain requirements, system specification requirements, mechanistic systems, romantic systems, user involvement.
Principles and Actions in Software Development	<ul style="list-style-type: none"> • Create a software development environment that is ontology driven consisting of a knowledge base, case generator, ontology editor and a reasoner. • Create an environment that allows developers to have

<p>Process that drive Interpretations.</p>	<p>sufficient time and resources to study the organizational culture, context and environment.</p> <ul style="list-style-type: none"> • Involve stakeholders, especially users and developers, during domain, business and specification modelling. • Actions: Develop a domain ontology, business ontology, method or process ontology for the system analysis model. • Create an ontology knowledge base for the system. • Create case designs for the different software modules. • Create or generate software codes from the ontology-driven case designs. • Reuse case designs from previous development processes and generate new cases.
---	--

The approach presented in Table 6.6 can be referred to as a family of methodologies. The methodologies found in this approach should, therefore, share the philosophy espoused in this approach, as well as the guiding principles. At this juncture, the research proposes a generic methodology that can be used with this approach. It should be noted, however, that the above approach can have an infinite number of methodologies. Using Benson and Standing’s (2005:203) characterization of a methodology as a grouping of “phases, procedures, rules, techniques, tools, documentation management and training for developers”, this study will move a step further and outline the structure of a methodology, as shown in Table 6.7 that can be applicable to this approach. This definition of a methodology has been adopted because, unlike other definitions given in this study, it clearly stipulates the categories whose sum total requires consideration in the development of a methodology.

Table 6.7: An Ontology-Driven Software Development Methodology

Ontological Software Development Methodology					
Phases	Procedures	Rules	Techniques	Tools	Documentation
Project Preparation	Create a software development platform. Develop or adopt a software development methodology ontology.	Ensure that the software development culture and context is studied.	Use software development methodologies ontology.	Ontology editor and ontology knowledge base, users and developers.	Ensure that documentation is captured in the ontology knowledge base (the repository).
System Study	Define business goals. Gather users’	Ensure that developers study the practice and	Interact with users or use the business and domain		

	wants, needs, goals and problems. Define the software requirements of the system.	contextual situatedness of the system. Create a language community among stakeholders.	ontology knowledge base. Choose an appropriate communication technique. Use the configuration management database.		
Software Requirements Study	Develop an analysis model. Develop a business model. Develop a domain model. Develop a specification model.	An ontological analysis model should be built to represent the software model that will interface with the design phase.	Discussion forums, joint application design sessions, etc. Domain, method, process or activity ontology development.	Ontology-driven case tool.	
Software Requirements Design	Choose the programming platform. Develop software case designs. Develop the software architecture.	Ensure that the ontological analysis model is transferred to the design phase.	Ontology design models, software case designs.	Ontology-driven case tool.	
Software Implementation	Build the software product.	Use an ontology-driven software programming platform.		Ontology-driven software development environment.	

The methodology shown in Table 6.7 is extremely generic and, at this predevelopment stage of the methodology, it is recommended that the reader does not focus on the techniques and tools listed. The software engineering field is vast and the techniques and tools listed here may not satisfy the requirements of some specific development environments. The researcher thus cautions the reader to start from the framework of Table 6.5 through to the approach of Table 6.6 before selecting any particular methodology.

6.11 Discussions

At this juncture, it is important to expand on the type of theory that has been generated in this study. Four frameworks were presented in this chapter: Table 6.4, the requirements framework, Table 6.5, the ontology-driven software development framework, Table 6.6, an ontology-driven approach to software development and Table 6.7, an ontology-driven software development methodology. It is important to find out the general type of theory represented by any of these frameworks. To achieve this, the classification by Gregor (2006) will be employed. As an interpretive type of study, a constructive type of theory was developed in this study in the form of frameworks. It is, therefore, important to find out whether each framework aims at analyzing and describing, explaining, predicting or prescribing, which are the four primary goals of theory as discussed by Gregor (2006).

Furthermore, it is important to check whether each framework consists of the necessary components of theory, that is, means of representation, constructs, and statements of relationships, scope and other issues that pertain to the purpose of the theory (Gregor, 2006). Using Gregor's (2006) taxonomy of theory types as a lens, it is noted that Table 6.4, the requirements framework falls under Type I theory. Its main focus is addressing the question "What is?" It is solely for analysis and description. What are the problems in software development? Table 6.5, the ontology-driven software development basically analyses and explains, addressing the "What is? Why? and How?" of the problem. It helped our understanding of the problems in software development and what we can do to address them.

A deep structure of ontologies was revealed and their characteristics were used to present a set of possible solutions (means) to these software development problems. While most of the software development problems listed in this framework are not new, the use of the listed ontologies to solve these development problems was not very obvious. Tables 6.6 and 6.7 can be categorized as both theories of explaining and predicting. The boundary between the various types of theory is not very clearly defined. As such, the most important tenet to be observed and judged in the theory is whether it further new insights and understandings in the field of study. There is no single clear answer to this question, as will be reflected in Section 7.3 of Chapter 7.

6.12 Summary

In this chapter four frameworks were developed that could be used to improve the software development process: the requirements framework, the ontology-driven software development framework, the ontology-driven software development approach and an accompanying methodology. Based on the grounding from which these are derived, they can be used to bridge the gap between mechanistic and romantic software development procedures. This chapter also supports the motivation for developers to concentrate on the relativistic paradigm and to assume behavioural development approaches during the development of software products.

Another valuable contribution of this chapter is the grouping together of many issues that bedevil the field of software development (*Table 6.4*). This list can be used as a quick reference guide to the problems and requirements that need consideration during software development.

Other aspects that should be recalled from Chapters 4 and 5 and from this chapter are that ontologies are computer processable and can capture the domain knowledge, business knowledge and specification knowledge of a system. By and large, they can, therefore, fill the language requirements gap discussed here. These frameworks need to be complemented with programming languages and should be upgraded to encompass ontologies in a bid to improve the implementation part of the development approach.

Lastly, since the ontology concept can be used to understand the meanings of concepts, an ontology knowledge base should be developed with each software development process, so that developers can always consult it and compare the different meanings of terms used by different stakeholders and organizations that are interested in different application domains. Chapter 7 is dedicated to a discussion of quality issues and of future work in this field of software development and ontologies.

CHAPTER SEVEN

RESEARCH EVALUATION, CONCLUDING STATEMENTS AND FUTURE WORK

7.0 Introduction

A research process is a journey that follows a very rough and winding road and is also filled with great uncertainty and suspicion. At each milestone, however, the research should reflect on the road to the deliverable and constantly check on its contribution to the overall goal of the research. At each point in the research process the researcher should check on whether the problem is being addressed, the research questions are answered and the quality criteria are met in order to contribute to the acceptance of the generated theory in a field of practice.

This chapter allows the researcher to step back and reflect on the completed research. Firstly, the quality considerations will be discussed. These are closely linked to the discussion in Chapter 3. Secondly, the contribution of the results of this research to the field of software development is discussed and, lastly, any unfinished business is highlighted.

7.1 Research Quality Considerations

As in all qualitative research tasks, process documentation is a very critical requirement of the study. The present researcher is of the opinion that the documentation and annotation of each process undertaken in the research process is the ultimate guide to ensuring quality and rigour in a qualitative study.

The quality of this research study, as discussed in Chapter 3, is based mainly on the work of Gasson (2003) and of Klein and Myers (1999), in which separate but related sets of criteria were proposed for judging and ensuring quality in qualitative researches. The discussion here is an integration and synthesis of the criteria from both of these sources.

In accepting the reflexive nature of a GTM research process, the researcher should ground the research process on the principle of the hermeneutic cycle. The finding from every GTM process should be based on a process of cyclic interpretation of data and repetitive execution of several mandatory steps. It should be noted that qualitative research is not about sample size but about rigour in the analysis of the qualitative data that are gathered. On this note, the data from interviews were analyzed on two separate occasions in order to check and ensure consistencies in the categories generated. At the same time, as suggested by Gasson (2003),

all the processes were documented in order to ensure the validity and reproducibility of the research results.

GTM reflexivity of the data-analysis process, as enshrined in the constant comparison method, is a good measure of the internal consistency of the research findings. This process ensures that the biases and distortions inherent in the data and the analyses processes are identified and minimized. In this study internal consistency was also read in conjunction with Klein and Myers' (1999) principle of suspicion. These two quality criteria are both tasked with identification of errors in the research process.

Every research task and problem has its uniqueness, situation and context. The quality of a GTM study will be enhanced if the context of the study is fully described. The nature of the study, the research process, the methods of data collection and the analysis and findings from this should be described comprehensively in the context of the research environment. This should be done to satisfy the principle of contextualization (Klein and Myers, 1999), in conjunction with confirmability and dependability requirements, as proposed by Gasson (2003).

The research covered the very specific study discipline of software development. The soundness of the results and of the theory generated relied heavily on the type, characteristics and on both the respondents' and researcher's knowledge. Many of the stakeholders in the field of software development are not necessarily software developers. Bearing this in mind, the researcher had to apply very strict sampling criteria in order to choose those respondents who had both generalist and specialist types of knowledge in this field.

As discussed in Section 6.3, open coding was done using interview scripts from respondents who had worked in industry and were both researchers and academics in the field of software development. This stratification, in particular, allowed the researcher to get open codes that ranged from the paradigm requirements of software development, development approach, methodological approach, communication approach to issues of adaptability and evolvability of software products. All these issues, supported by the study of literature (Chapters 4 and 5), highlighted the problems that are currently not being addressed in the field of software development.

Since the context determines the nature of the results generated from the research study, the sampling used in this research allowed the researcher to have an open view of the issues that

needed to be addressed in the software development discipline, before finalizing the research questions and the propositions addressed by the research.

After finalizing the propositions and research questions and doing selective coding and trying to reach theoretical saturation, the researcher focused the context of the research on project managers, software developers and software testers, in order to pick specific issues that would answer the research questions elicited. This also allowed the research to be focused and not to go astray.

Lastly, the GTM findings should be transferable between contexts. The task here is for the researcher to move from substantive theory to formal theory (Glaser & Strauss, 1967; Strauss & Corbin, 1990). Through research in different contexts or the involvement of different researchers in the same substantive theory, generated theory can be generalized and adapted to apply to different scenarios. This would ensure that, in addition to the existence of multiple interpretations, the formalized theory would take into consideration the varied assumptions, interpretations and work settings used by different researchers. This process would increase the transferability of the research results.

In this research, the ontology-driven framework that was developed by mapping M(2) was further theorized to come up with the ontology-driven approach and methodology. This satisfied the principle of multiple interpretations. Consequently, the acceptability of the knowledge cannot be doubted to any great extent, but the findings can be applied in real life. The ontology-driven software development approach and its accompanying methodology are at a higher level of formalization than the ontology-driven software development framework.

7.2 Contribution to the Field of Software Development

In the context of this research, it is of paramount importance that the theoretical aspects or contributions to the body of knowledge be identified. At the same time, the theoretical aspects or contributions should be balanced with the pragmatic aspects of their applicability in a social setting. This process is gradual and systematic but will finally lead to a paradigm shift, a notion that the author is aiming for as a slow but salient objective of this research.

The theory developed in any research requires an actor if it is to impose itself in a social practice. Agerfalk (2004) defines an actor as an artefact, an abstract or concrete thing that is able to interact in an organizational setting. The actor should, therefore, work as a change agent, thereby changing the way things are done in the recipient organization.

The contribution of this research is multifaceted. As a starting point, the research contributed to the philosophical discipline of information systems (Chapter 2), information systems research methodologies (Chapter 3) and the theoretical underpinnings of research and practice in information systems. This contribution is reflected in the research publications listed in Appendix B. These papers discussed the importance of using ontologies in software development (Mavetera, 2004a), the philosophy of ontologies and their role as obligatory passage points in information systems (Mavetera, 2004b), the central role that ontologies can play in software development process as communicative and productive agents (Mavetera, 2007), the issues that software developers deal with in their everyday lives (Mavetera & Kroeze, 2009a, 2009b), the practical revelations in using GTM in qualitative research (Mavetera & Kroeze, 2008, 2009c, 2010a); and the ontology-driven software development framework were discussed by Mavetera and Kroeze (2010b). The theoretical and methodological groundings presented in Chapter 2 have been published as a book chapter (Mavetera, 2010) and, lastly, the whole research focus of enriching information systems with humanities was discussed at ECIS 2010 panel discussion and is found in Kroeze *et al.* (2010). This was further expanded in Kroeze *et al.* (2011) and finally the ontology-driven software development approach was discussed in Mavetera (2011). It is the researcher's intention to see that these outputs contribute to the practice of both software and information systems research and development.

As discussed in Section 2.1, the research has built up a body of scholarly work in the fields of software development and information systems, as well as in the fields of research philosophy and methodologies. This scholarly contribution, as discussed by Cornford and Smithson (2005), has added new knowledge in these specific study disciplines.

As Kuhn noted, scholarly work is judged by sound theoretical underpinnings (Chapters 2 to 5) that exist in a stable paradigm which is clear to people who share the same thinking in a field of study. These theoretical underpinnings exist also as theoretical frameworks (AT, ANT, TOA, TOC, Grounded Theory and sociological paradigms); upon which all actions performed in social practices can be soundly be based (Burrell & Morgan, 1979).

Guided by these theoretical underpinnings, the major output of this research was presented as an ontology-driven software development approach for use during software development. This came as result of continuous investigation or research into these scholarly fields and resulted in changes to some accepted theories in software development. New ideas and softer

methods of developing software products, as proposed here, may gain authority and possibly result in the rejection of old ideas.

Also, as discussed by Olivier (2004), the three strategies (*see Section 2.2*) that can be used to conduct research have been satisfied by this research. For completeness, the strategies state that research can be carried out to:

- a) Compile information on a topic whose bits and pieces have already been discovered (often by other researchers), but which have not yet been integrated into a single coherent body of knowledge.

Much research has been carried out in the use of ontologies in information systems and in software development in particular, but most of this has been presented as disparate pieces of information addressing varying requirements or components of issues in software development practice. The present research has, to a large extent, combined these knowledge areas, the sum total being a holistic framework that improves the practice of software development.

In addition, practitioners in the field of software development have long discussed the need for observing culture, context and pragmatics but, to the researcher's knowledge, these aspects have not been compiled into a single body of knowledge that portrays cause, effect and solutions. An attempt to do this was made by Roque *et al.* (2003) but a framework for the implementation of these issues in software products was not developed. To a great extent Chapters 1 to 6 of this study managed to bring this body of knowledge under a single umbrella.

- b) Look with new eyes at existing knowledge (standard ways of doing things) and to trying to find a better solution for a problem that has previously been solved and, lastly,
- c) To solve a problem for which no known (or apparent) solution exists.

From the refined research questions and propositions discussed in Sections 6.5 and 6.6 respectively, this research addressed all these three strategies to some extent. The research integrated bits and pieces of knowledge that can be added to the software development body

of knowledge and to the ontology research body of knowledge. It also addressed the persistent problems in the mechanistic development of software products.

7.3 Evaluation of the Study's Contribution to the Body of Knowledge

A PhD thesis can be regarded as the final output of a long, systematic process of engaging in focussed research. Regardless of the processes followed, a research study should establish a research question in a specific study discipline. For it to be systematic, the research study must be guided by a research methodology and at the same time be grounded in some theoretical underpinnings that are used as lenses to unravel what exists and can be known (ontology and epistemology respectively) in a particular field of study. Also, as a thesis, the execution and the reporting process of these tasks must be serial, with related arguments building one on top of the other. This must lead to theory generation or to its refutation.

The most important aspect is to assess whether the research study has made some progress or not (Introna, 1992) to the addition of knowledge in the field under study. As a dictum, Introna (1992) urges researchers not to appraise a theory in isolation. By this, we note that the entire contents of the thesis should be considered, with particular attention being paid to the way in which arguments were raised and synthesized. A substantial contribution to knowledge in a field of study should not be used as the sole criterion and should not be a priority.

As noted by Introna (1992:5-30), assessing “one’s own theory may prove to be very difficult indeed”. He further urges readers to remember that the result of the evaluation process is less valuable than the process of carrying out and documenting the research study in the form of a thesis. It is in these processes that learning and understanding are achieved. Understanding in this context can be likened to the individual’s development of sufficient “descriptions of the conceptual structures” of a discipline, its relationship to other related disciplines and ethical practice thereof (Green, 1998:176).

Quite often, especially in positivist studies, falsification is used to judge the validity of a theory. However, this same falsification cannot be used as a way to verify the validity of a grounded theory (Suddaby, 2006). Instead, the constant comparative method, a process of continuously evaluating “emerging constructs against ongoing observations” is used to test the emerging conceptual structures (Suddaby, 2006:636). Furthermore, although grounded theory can reflect itself in many forms, it manifests itself more often as narrative scripts (Trim and Lee, 2004). Trim and Lee (2004) further warn researchers to choose their

representation of GTM carefully as it is the basis (grounding) upon which the final theory is judged and accepted. The survival of the grounded theory is heavily dependent upon a complete and holistic explanation of the data. This is closely intertwined with how extensively and accurately grounded theory permits observers to identify and predict connections between both conceptual and practical aspects.

7.3.1 Evaluation of the Generated Theory

The evaluation criteria used here was established in Section 3.11. It will now be used to evaluate the theory generated by this research. Some of the criteria used in evaluating the contribution of the thesis are:

a. Is the research problem addressed a persisting question in the field?

There have been several research studies in the areas of software development. Some focused on improving requirements gathering, on developing new development techniques and new process models, to mention but a few. In short, the software development problem, coined as the software crisis, has already persisted for several decades and will possibly continue to persist for decades to come. The research problem addressed in this study focused on improving the quality of software products developed and especially on how human aspects can be included in software products. These problems boil down to the whole methodological debate on software development, of which there are many variants of methods, to which this research study has added one more. In short, the research did not fall short of proposing a novel theoretical or methodological slant to software product development. From Section 6.5, the following research questions that guided the development of the theory have to be checked to determine whether they were addressed. It must be noted that research questions presented in Chapter 1 are no longer going to be considered because they have been superseded by these refined questions.

- i. What are the problems within organizational information systems that inhibit their usability?*

This question touched on the mechanistic nature of current organizational information systems, and blamed most of the usability problems with the systems on the way in which they were developed. The systems, as discussed in Chapter 4, do not capture

softer elements of organizational systems, such as culture, meanings of the concepts used or the context of the system, amongst others.

- ii. *What are the problems that are persistent and currently experienced during software development?*

The overarching issues here were found to be the development approaches currently in use. Three development approaches, that are at times misunderstood, the structured, object-oriented and agile approaches, were blamed for using methods, techniques and tools, all of which fall in the hard-systems paradigm. Also, developers' reliance on systematicity and reductionism has compounded the problems. However, on a finer granular level, the persisting software development problems have been identified as communication problems, the lack of domain and business knowledge in the analysis, design and implementation models, and the prescriptive nature of all the models that are used to implement current information systems. These were addressed in detail in Chapters 1, 4 and 6 of this study.

- iii. *What are the requirements of a software development approach that should address these current problems in software development?*

This question was progressively addressed from Chapters 1 to 6. Chapter 2 went as far as discussing the paradigms that should guide the choice of development approaches for software development. Section 4.8.4 to 4.10 in Chapter 4 used literature findings to explain what has been said about software development approach requirements. In Section 6.8 these were combined with empirical data obtained from interviewees to provide a comprehensive requirements framework for a software development approach.

- iv. *What are the software development problems that can be addressed using ontologies?*

As this was not a very direct question, direct answers could not be provided. The research, however, after eliciting the problems currently bedevilling the software development process, focussed on the discussion of ontologies, that is, on the What?, Why" and How? of these ontologies. This was pre-empted in Chapter 1, and extensively covered in Chapter 5. Using the Double Mapping Principle developed in Section 3.5 of Chapter 3, the software development problems were mapped to the

respective ontologies that can be used to address them in Sections 6.7 through to 6.9 of Chapter 6.

- v. *What are the characteristics of ontologies that are important and can be used to represent organizational information systems?*

This question was answered in Chapter 5, in which the essence of information systems ontology was discussed. Although the architectural aspects of ontologies and the concept of conceptualization were touched upon, the most important issues addressed were the ability of ontologies to capture semantics, pragmatics and social context, among other aspects such as syntactics. These have been used to “sell” the idea of a romantic information system model.

- vi. *What are the ontology characteristics and ontology varieties that can be used to develop an ontology-driven software development approach (ODSDA)?*

Having discussed the ontology characteristics in Chapter 5, several ontology varieties have been presented, such as domain, method, social, process, intentional and status ontologies. These are not really exhaustive, but are very important in addressing the requirements of the thesis, that is, developing a software development approach that can capture and maintain the softer human elements of organizations in the software products. These ontology varieties have been included as components of the ODSDA presented in Chapter 6. To a much greater extent, the research questions have all been addressed in the thesis. More importantly, these enabled the theorizing of the ontology-driven software development approach and methodology presented in, Section 6.10 of Chapter 6 and also reflected in Tables 6.6 and 6.7.

b. Is the method of enquiry adopted suitable?

There has to be a systematic and appropriate application of the research method, in allowing corroboration or refutation of the results by other scientists. The method of enquiry cannot be judged separately from the theoretical underpinnings and methodological underpinnings of the research. Thus, before the GTM was chosen as the method of enquiry, it was essential to include the discussions in Chapter 2, covering the ontological, epistemological and humanist stances taken about the nature of reality. These underpinnings can be considered as a lens through which the research problem is

observed. On the methodological grounding, the researcher must clearly document all the steps taken, giving reasons for data-gathering, analysis and presentation of the results. Chapter 3 even allowed a slight diversion to enable other research methods that could be possible candidates of the research enquiry to be discussed. This allowed the researcher to build enough ground to support the chosen research approach. Since the research aimed at constructing a theoretical framework for use in software development as a constructivist method, GTM became the method of choice. As discussed earlier, GTM, in addition to building theory from the field data, it also uses secondary data in the form of existing literature to supplement the field data.

c. Is the method for data analysis chosen appropriate and aligned to the method chosen?

This is quite critical in qualitative studies such as the one reported here, where GTM, in particular, is used. There are many dicta proposed by Glaser and Strauss (1967) at its inception that have been continuously adopted with several amendments. It is of paramount importance that each step in the analysis be described in the context of the research. The necessity for this is further accentuated by the fact that many of the steps were executed manually and that documentation of these steps followed precisely the way in which these manual activities were carried out. With the advent of automated analysis tools, such as Atlas Ti used in this study, the documentation does not necessarily show how the analysis was done. Typical examples are the processes of axial coding and constant comparative methods whose products are all housed in the software repository but whose presentation in hard copy form may require several pages. It is, therefore, in the spirit of technological advancement that these are included in the appendices as soft copies and that only the findings are reported in the main document. This is heavily supported by University of Victoria (n.d.) which require only important summary data findings to be included in the main body of the research report, relegating the rest of primary and secondary data to the appendices. Several GTM aspects were followed and it is quite important to list in the data analysis where these principles were followed.

d. Has the theory unified various, previously unrelated problems or concepts (Introna, 1992)?

Prior to this study, as indicated in Chapters 4 and 5, many organizational concepts and problems were treated disparately. These included, for example, problems of communication in software development, capturing of requirements, composition of the analysis model, its mapping to the design model and implementation model, as well as problems of capturing software issues in organizational systems such as culture, context, semantics, pragmatics, to mention but a few. The systematic introduction and use of ontologies in software development has also been a persistent problem. The theory can at least be seen as a moderate advancement towards a lasting solution to most of these problems. So, yes, to a greater extent than previously, this theory has unified various, previously unrelated problems.

e. Has the theory produced new perspectives on existing problems? This should lead to a new understanding of the persisting problems under investigation. (Introna, 1992).

The problem addressed in this study is not very simple as many a reader will think. The research did not address the well known issues of why organizational systems persistently fail to provide return on investments, why software products persistently fail to capture softer human aspects of organizations and the like, or the simplistic question of what are the problems encountered during software development at the requirements-gathering, analysis, design and implementation stages were not the core of the problem either. Many issues were partly touched on in the thesis that may divert the attention of the reader away from the main purpose of this study. Ultimately, the research addressed the problem of how information systems could be more representative of the socially constructed organizational systems that they are supposed to represent. This has to be addressed from the time that software products that are used later to build information systems are developed. Therefore, an approach that could improve the capturing of these ever-elusive human attributes, and obviously encompassing a methodology, had to be devised, which Kroeze (2009) and Kroeze *et al.* (2010) termed '*Humanities-enriched Information Systems*'.

Based on the new perspective generated on existing problems, some of the software development problems now have answers, for example including but not limited to the following questions:

- i. What are the semiotic components of ontologies?
- ii. How can organizational human attributes of culture and context be captured and maintained in information systems?
- iii. How can ontologies be systematically introduced into software development and information systems?

f. Has the theory produced unconventional ideas?

As discussed in Section 3.11, these ideas could be radical and challenging to the existing beliefs in the field of study (Introna, 1992). Using Stokes and Bird's (2008) minimal creativity criterion as a yard stick, two conditions that is, novelty and agency, must be checked at the minimum.

Ontologies have been used in information systems, but no one had ever tried to design an approach for using ontologies in software development. This is likened to a software development process model. This is a moderate measure of historical novelty that is judged using "socio-technical facts, not behavioural ones" (Stokes and Bird, 2008:230). On the same note, had this research not explored the two fields of software development and ontologies, elicited software development problems associated with capturing human aspects of organizations, mapped these problems to ontology characteristics that could address them and finally come up with an ontology-driven software development framework, then surely an ontology-driven software development approach depicted here could not have been achieved. This is also a relatively good measure of agency as described in Section 3.11, that is "some behaviour, artefact or event **F** is the product of the agency **A** only if **F** would not have occurred had **A** not acted in some autonomous way" (Stokes and Bird, 2008:231). The sum total of these two, as portrayed in this thesis, is that, a moderate degree of creativity has been achieved.

g. The theory should exhibit positive and negative heuristic power (Introna, 1992).

This can be regarded as the ability to assist developers in solving software development problems. The negative heuristic power can be viewed as the following:

- i. There is no need for a new software development approach or methodology. The current approaches are sufficient.
- ii. The existing ontology applications are well understood and are sufficiently annotated in the information systems field.
- iii. The softer aspects of organizational systems cannot be captured and maintained in software products.

The positive heuristic powers may be regarded as follows:

- i. There is great potential for improving the implementation of information systems by including semantics, pragmatics and context in the software product.
- ii. Ontologies greatly improve the mapping of the business model, domain model, specification model and the analysis model to the design model through to the implementation model.
- iii. Ontologies used in software development are still very much a philosophy, and need to be operationalized at a more practical level so as to add value in software development. This also applies to the ontology-driven software development approach presented here.

h. Is there a sufficient study of the relevant literature?

It should be noted that a research study that culminates in a thesis must be grounded in a sound study of literature, with the emphasis on recent developments in the substantive areas of study. Judging from the nature of the substantive discipline of the study, as well as of software development and information systems, it is difficult to quantify the extent to which one can rate a sound study of literature. This is possibly one of the oldest and most researched aspects in the field of Information Technology. In addition, it is also quite diverse. Its diversity can be measured by the different emphasis placed by researchers from different continents on what is currently important in the field of software development. Research trends in Europe, America, Asia and Australasia, although grounded upon some very similar tenets, always tend to follow and emphasise on separate issues. On that note, any literature coverage for an academic research study can hardly be deemed sufficient. This is exacerbated by the fact that two substantive areas were consulted in the study: ontologies and software development. The strands followed could mislead our judgement, particularly since these fields may follow a management, behavioural science perspective or a technical, design science perspective.

The safest position to assume in such a debate would be to accept that literature study can never be exhaustive.

On another angle, this could be judged from the point that each research problem has specific literature that may be quite relevant to its universe of discourse. The questions of what the research problem is, why it is a problem and of how it should be addressed are pertinent questions that may direct the literature study. In the end, availability of literature sources and time constraints may well dictate the access to these sources. It should be noted, however, that it is important to acknowledge different academic discourses and possible contradictions in views, both theoretical and methodological.

7.3.2 Evaluation

Akhlaghpour *et al.* (2009) argue that there exists a salient problem in IT research. This problem has since compelled IS researchers to choose more conservative [research] topics. However, this attitude has seriously deprived the generation of novel ideas in the IS field. This may be attributed to the need for any IS research to be considered “legitimate” in the eyes of a group of individual scholars. More often than not, these scholars possess different interpretations of what is being researched and is researchable and acceptable in the core discipline of IS. This also adds to an ever-growing number of varied interpretations from IS scholars, which are dictated by their varied identities, depending on their countries or continents of origin (Akhlaghpour *et al.*, 2009). If that were not enough, IS researchers can be accused of mechanistically adopting theories from reference disciplines, of preferring to use well developed theories from other social sciences and then forcing the IT artefact investigation to conform to it.

From the evaluation presented in this chapter, the reader may be convinced that there has been indeed scientific progress in the field of software development, methodologies and ontologies. This, however, brings us to the second part of the title of this thesis, “An unended quest”. This research only provided guiding principles, a framework *per se*, that can be used to start the development of design artefacts that are needed to realise the dream of capturing and translating human behavioural attributes into information systems. This is just the start of the beginning. The questions that remain unanswered, therefore, include but are not limited to:

- a. What are the IS and ontology development platforms that should be used to develop the ontologies prescribed in this approach?
- b. How do we address the unavoidable failure to capture tacit IS human issues such as feelings?
- c. What are the implications of this new development approach to the field of software development practice?
- d. What are the constituents of human behaviour from an IS perspective?
- e. What are the implications of this study results to the design of IS curriculum?

Obviously, there are still many unanswered questions in this research study. However, if new knowledge is to be created, then different methodological approaches, such as the double-mapping principle used during data analysis and the ontology-driven software development approach discussed here must be adopted so as to provide a new lens for looking at problems and reality (Trim & Lee, 2004). This also answers Wyssusek's (2004) call for IS practitioners to find more rigorous theoretical foundations upon which they can base their IS development activities. As artefacts built from the behavioural and technical aspects of organizations, a social constructionist view must be adopted and ontologies are having high applicability in this field.

7.4 Recommendations, Future Work and Limitations

The fields of software development and ontologies are scientific disciplines that are always affected by their situation in a social setting. As the environment for these two disciplines is always in a state of flux, the methodologies used in this practice should always be improved upon on a continuous and regular basis.

The research requires more work to be done on the application of this approach in work settings. The results obtained should be judged using the quality, cost, timelines and ability to capture the human elements of organizational systems as criteria. Also, researchers should develop a software engineering environment that addresses these humanist elements and is ontologically driven, as proposed by Mavetera (2007) and described in Section 5.8.

There are several research studies that need to be completed and which could contribute significantly to the final acceptance, diffusion and infusion of these research results in the software development industry. These studies include but are not limited to:

- Development of the ontology of information systems development methodologies (ISDM). A general ISDM ontology was developed by the author's student (Shawa, 2009) as part of the requirements for the degree, Master of Science in Computer Science and Information Systems. However this needs to be done at an industrial scale in order to produce a full ontology that could be used during software development.
- Examination of the ontological software development methodology discussed in Section 6.10 indicates that it is critically important that a method of arriving at the deliverables at each phase be developed. For example, a method for developing domain, business and process ontologies to capture the domain, business and specification models into the analysis model should be developed.

All these requirements could be used to validate the framework developed in this research. However, the scope of the research study did not allow the researcher to accomplish this task. One way of accomplishing this is to team up with industry partners who can use this framework in the development of software products. As this process would take a lot of time and the time allocated for a Ph.D. study does not allow this, it was decided that these tasks be deferred to future research activities. It is important to note that GTM, as proposed by Glaser and Strauss (1967), does not require researchers to validate or verify the theory so generated. As Glaser (1992:31-32) stated, the generated theory “need not be verified, validated, or more reliable”.

There are several issues that can be likened to the limitations of this research study. For example, during the primary data gathering process, all of the interviewees had no knowledge of IS ontologies. This posed a big challenge because they could not relate software development methodology issues to the ontologies characteristics. This problem accounts for the lack of any ontology specific questions in the interview questions. This challenge was circumvented by introducing a third tier in the analysis framework in order to link problems in IS development to what ontologies can address. This is reflected in the expanded double mapping principle of Figure 6.2, where categories developed using M1 was mapped to M2. This second mapping could however introduce some distortions to the final results.

Another issue is that constructivist research does not rely solely on the empirical support of the theory generated. However, empirical evidence can be minimally used to support the

building up (theoretically) of the theory which may have mainly either practical or epistemic utility or both. This stance usually poses challenges in the scientific domain in trying to convince people to accept the theory so generated. Another limitation for this research is its constructivist nature. This made most of the arguments and results from these arguments very theoretical, hence it's so called abstractness. Due to its requirements of developing a software development approach, no time was dedicated to the provision of a 'proof of concept'. This should have been realised as the use of this approach in a real life development of software products. While the process is very lengthy, the reader would have been more comfortable seeing examples of the domain, business, and intentional, process, social ontologies mentioned herein. Also, the ontology driven SDE prototype could actually help to sell the idea of this ontology-driven software development approach. However, this is still 'Towards an ontology-driven software development approach: an unended quest'. Future work will address most of these requirements.

7.5 Conclusion

This research should be viewed as the start of a series of research projects that are compiling a body of knowledge that can be used in the development of romantic software products. In order to appreciate the contribution of this study to the software development body of knowledge, the reader should realize that new paradigms take a long time to be accepted. This research calls for a paradigm shift and for acceptance of the fact that current development methodologies and approaches are mechanistic. A transition is, therefore, required to accept the virtues of the romantic software development paradigm proposed in this study. The reader should also note that, at each discussion point, the limitations of the research methodology used in this study were discussed.

BIBLIOGRAPHY

- AGENTIS, n.d. *Simplifying the complexity of application development. Developing and deploying J2EE solutions with Agentis Adaptive Enterprise TM Solution Suite*. Atlanta, GA: Agentis International, Inc.
- AGERFALK, P.J. 2004. Grounding through operationalization. Constructing tangible theory in IS Research. *Proceedings of the 12th European Conference on IS (ECIS 2004), Turku, Finland, 14-16 June*.
- AKHLAGHPOUR, S., WU, J., LAPOINTE, L. & PINSONNEAULT, A. 2009. Re-examining the status of “IT” in IT research - An update on Orlikowski and Iacono (2001). *Americas Conference on Information Systems (AMCIS) Proceedings, San Francisco, California, August 6-9, pp. 1-14*.
- ALEXANDER, P.M. 2002. *Towards reconstructing meaning when text is communicated electronically*. Pretoria: University of Pretoria (PhD Thesis, University of Pretoria). [Online]. Available: <http://upetd.up.ac.za/thesis/available/etd-08192002-155431/> [Cited 10 May 2006].
- AVISON, D. & FITZGERALD, G. 2006. *Information systems development: Methodologies, techniques and tools*. Maidenhead, UK: McGraw Hill.
- AVISON, D.E. & FITZGERALD, G. 1995. *Information systems development: Methodologies, techniques and tools*. 2nd ed. Maidenhead: McGraw Hill.
- AVISON, D.E., DWIVEDI, Y.K., FITZGERALD, G. & DOWELL, P. 2008. The beginnings of a new era: Time to reflect on 17 years of the ISJ. *Information Systems Journal*, vol. 18, no. 1, pp. 5-21.
- ABMANN, U., ZSCHALER, S. & WAGNER, G. 2006. Ontologies. Meta-models and the model driven paradigm. In *Ontologies for Software Engineering and Software Technology* (Coral Calero, Fransisco Ruiz and Mario Piattini, eds.). Berlin: Springer Verlag, pp. 249-274.
- BAKER, R. 2006. *What software crisis?* [Online]. Available: tech.norable.com/2006/05/what-software-crisis.html [Cited 3 November 2008].

BASDEN, A. 2001. Christian philosophy and Information Systems. *Presented to Institute for Christian studies, Toronto*. [Online]. Available:
<http://www.isi.salford.ac.uk/dooy/papers/cpis.html> [Cited on 31 August 2006].

BENSON, S. & STANDING, C. 2005. *Information Systems: A business approach*, 2nd ed. Australia: Wiley & Sons.

BENSTA, H. 2010. Conceptualization of a framework based on ontology for construction an object model. *Proceedings of the IBIMA 2010 conference: Business transformation through innovation and knowledge management: An academic perspective, Istanbul, Turkey, June 23-24*, pp. 2058-2070.

BERG, B.C. 2007. *Qualitative research methods for the social sciences*. 6th ed. Boston, MA: Pearson Education Inc.

BEYNON, M., BOYATE, R. & CHAN, Z.E. 2008. Intuition in software development revisited. *Proceedings of the 20th Annual Psychology of Programming Interest Group Conference, Lancaster University, UK, September 10-12*.

BHASKAR, R. 1989. *The possibility of naturalism: A philosophical critique of the contemporary human sciences*, 2nd ed. Hemel Hempstead: Harvester Wheatsheaf.

BJØRNER, D. 2008. *Software engineering: An unended quest*. Denmark: Technology University of Denmark (Doctor of Technology Thesis, Technology University of Denmark).

BROOKS, F.P. 1987. *No silver bullet. Essence and accidents of software engineering*. Computer Magazine. [Online]. Available:
<http://virtualschool.edu/mon/softwareEngineering/BrooksNoSilverBullet.html> [Cited 10 June 2008].

BROOKS, F.P. 1995. *The mythical man-month*. Anniversary ed. New York, NY: Addison-Wesley.

BROWN, I. 2008. Investigating the impact of the external environment on strategic information systems planning: A qualitative inquiry. *Proceedings of the 2008 annual research conference of the South African Institute of Computer Scientists and Information Technologists on IT research in developing countries: Riding the wave of technology, Garden*

Route, Wilderness, South Africa, October 6-8, ACM International Conference Proceeding Series, vol. 338, pp. 8-15.

BROWN, R., NERUR, S. & SLINKMAN, C. 2004. The philosophical shifts in software development. *Proceedings of the Tenth Americas Conference on Information Systems, New York, August 5-8*.

BROWN, A.W., WALLNAU, K.C. & FEILER, P.H. 1991. Understanding integration in a software development environment: Issues and illustrations. *Journal of Systems Integration*, vol. 3, no. 3-4, pp. 303-329.

BRYL, V., GIORGINI, P. & MYLOPOULOS, J. 2009. Designing socio-technical systems: From stakeholder goals to social networks. *Requirements Engineering Journal*, vol. 14, no. 1 pp. 47-70.

BRYMAN, A. 2004. *Social research methods*. 2nd ed. Oxford: Oxford University Press.

BUITELAAR, P., CIMIANO, P. & MAGNINI, B. 2003. Ontology learning from the text: An Overview. In *Ontology Learning from Text: Methods, Applications and Evaluation*. Edited by Buitelaar, P., Cimiano, P. and Magnini B.

BURRELL, G. & MORGAN, G. 1979. *Sociological paradigms and organisational analysis*. London: Heinemann.

CALERO, C., RUIZ, F. & PIATTINI, M. 2006. *Ontologies for software engineering and software technology*. London: Springer Verlag.

CAMPBELL, B. 2009. A resolution of student's grounded theory a priori reading dilemma. *Proceedings of Conf-IRM, AI-Ain, UAE, May 21-23*.

CARNEGIE MELON SOFTWARE ENGINEERING INSTITUTE. n.d. *Software product lines*. [Online]. Available: <http://www.sei.cmu.edu/productlines/> [Cited 14 August 2006].

CHARMAZ, K. 2000. Grounded Theory: Objectivist and constructivist methods. In *Norman K. Denzin & Yvonna S. Lincoln (eds.), Handbook of Qualitative Research*, 2nd ed., pp. 509-535. Thousand Oaks, CA: Sage.

CHARMAZ, K. 2006. *Constructing Grounded Theory: A practical guide through qualitative analysis*. Thousand Oaks, CA: Sage.

CHECKLAND, P. 1999. *Systems thinking, systems practice: Includes a 30-year retrospective*. Chichester: Wiley.

CORCHO, O., FERNANDEZ-LOPEZ, M. & GOMEZ-PEREZ, M. 2006. Ontological engineering: Principles, methods, tools and languages. In *Ontologies for Software Engineering and Software Technology*. Edited by Coral Calero, Fransisco Ruiz and Mario Piattini. Berlin: Springer Verlag.

CORDEIRO, J. & FILIPE, J. n.d. *Language action perspective. Organizational semantics and the theory of organized activity. A comparison*. [Online]. Available: http://ltodi.est.ips.pt/jcordeiro/documents/demo2003CordFil_Final.pdf [Cited 4 May 2008].

CORNFORD, T. & SMITHSON, S. 1996. *Project research in Information Systems: A student guide*. New York, NY: Palgrave.

CRETU, L.G. 2010. Business process oriented software engineering. *Proceedings of the IBIMA 2010 conference: Business transformation through innovation and knowledge management: An academic perspective, Istanbul, Turkey, 23-24 June 2010*, pp. 1773-1786.

DAHLBOM, B. & MATHIASSEN, L. 1997. The future of our profession. *Communications of the ACM*, vol. 40, no. 6, pp. 499-579.

DAHLBOM, B. & MATHIASSEN, L. 1992. *Systems development philosophy. Computers and society*. ACM SIGCAS Computers and Society, vol. 22, no. 1-4, 1992.

DAHLBOM, B. & MATHIASSEN, L. 1993. *Computers in context. The philosophy and practice of systems design*. Oxford: NCC Blackwell.

DAHLBOM, B. 1996. The new Informatics. *Scandinavian Journal of Information Systems*, vol. 8, no. 2, pp. 29-48.

DE OLIVEIRA, K.M., VILLELA, K., ROCHA, A.R. & TRAVASSOS, G.H. 2006. Use of ontologies in software development environments. In *Ontologies for software engineering and software technology*. Edited by Coral Calero, Fransisco Ruiz and Mario Piattini (eds.). Berlin: Springer Verlag.

- DENNIS, A., WIXON, B.H. & TEGARDEN, D. 2002. *Systems analysis and design: An object oriented approach with UML*. New York, NY: Wiley.
- DICTIONARY.COM. n.d. *Theory*. [Online]. Available:
<http://dictionary.reference.com/browse/theory/> [Cited 27 June 2010].
- DILLEY, P. 1999. Queer theory: Under construction. *QSE: International Journal of Qualitative Studies in Education*, vol. 12, no. 5, pp. 457-472.
- DING, Y. & FOO, F. 2002. Ontology research and development. Part 1- a review of ontology generation. *Journal of Information Science*, vol. 28, no. 2, pp. 123-136.
- DITTRICH, Y. & SESTOFT, P. 2005. *Designing evolvable software products: Coordinating the evolution of different layers in kernel based software products*. [Online]. Available:
<http://www.itu.dk/research/sdg/doku.php> [Cited October 2006].
- DOBING, B. & PARSONS, J. 2007. What practitioners are saying about the Unified Modelling Language. In *Managing Worldwide Operations and Communications with Information Technology*, Information Resources Management Association International Conference, Vancouver, British Columbia, Canada, May 19-23, pp. 123-126.
- DRISTAS, S., GYMNOPOULOS, L., KARYDA, M., BALOPOULOS, T., KOKOLAKIS, S., LAMBRINOUDAKIS, C. & GRITZALIS, S. 2005. Employing ontologies for the development of security critical applications: The secure e-poll paradigm. In *Challenges of Expanding Internet: E-Commerce, E-Business, and E-Government*, IFIP International Federation for Information Processing, vol. 189, pp. 187-20.
- DU PLOOY, N.F. 2004. *Information technology change management*. Pretoria:University of Pretoria (class notes), South Africa, 2004.
- EL BAZE, N. 2005. Cracking software development complexity: In order to reap the benefits of progress in processor technology, we must fundamentally rethink how to write software. *Line 56-The Business Executive daily*. [Online], Available:
http://www.partechvc.com/newspdfs/Line56_NEB.pdf [Cited 30 March 2005].
- ENGINEERS COUNCIL FOR PROFESSIONAL DEVELOPMENT (ECPD). [Online] Available: <http://www.abet.org/history> [Cited on 11 January 2008].

FALBO, R.A., GUIZZARDI, G. & DUARTE, K.C. 2002. An ontological approach to domain engineering. *International Conference of Software Engineering and Knowledge Engineering, SEKE'02, Ischia, Italy, July 15-19.*

FENSEL, D. 2004. *Ontologies: A silver bullet for knowledge management and electronic commerce*, 2nd edition. Berlin: Springer Verlag.

FITZGERALD, B. & HOWCROFT, D., 1998. Competing dichotomies in IS research and possible strategies for resolution. *ICIS '98 Proceedings of the International Conference on Information systems (ICIS '98), Atlanta, GA: Association for Information Systems, Helsinki*, pp. 155-164.

FUENTES-FERNÁNDEZ, R. & GOMEZ- SANZ, J.J. 2009. Understanding the human context in requirements engineering. *Requirements Engineering Journal*, vol. 14, Springer-Verlag.

GARCIA-DUQUE, J., PAZOS-ARIAS, J.J., LOPEZ-NORES, M., BLANCO-FERNANDEZ, Y., FERNANDEZ-VILAS, A., DIAZ-REDONDO, R.P., RAMOS-CABRER, M. & GIL-SOLLA, A. 2009. Methodologies to evolve formal specifications through refinement and retrenchment in an analysis-revision cycle. *Requirements Engineering Journal* , vol. 14, no. 3, pp. 129-153.

GASCHE, R. 1986. Infrastructures and systematicity. In *Deconstruction and philosophy: The texts of Jacques Derrida*, edited by John Sallis, pp. 3-20.

GASSON, S. 2003. Rigor in grounded theory research: An interpretive perspective on generating theory from qualitative field studies. In *The Handbook of Information Systems Research*, edited by M.E. Whitman & A.B. Wozzczyński. Hershey, PA: Idea Group Publishing.

GERBER, A.J. 2006. *Towards a comprehensive functional layered architecture for the semantic web*. Pretoria: UNISA (PhD thesis in Computer Science, UNISA).

GIDDENS, A. 1976. *New rules of sociological method: A positive critique of interpretative sociologies*. London: Hutchinson.

GILB, T. 1985. Evolutionary delivery versus the "waterfall model". *ACM SIGSOFT Software Engineering Notes*, July 1985, vol.10, no. 3, pp. 49-61, July 1985.

GLASER, B.G. & STRAUSS, A.L. 1965. *Awareness of dying*, Chicago, IL: Aldine.

GLASER, B.G. & STRAUSS, A.L. 1967. *The discovery of Grounded Theory: Strategies for qualitative research*. New York, NY: Aldine De Gruyter.

GLASER, B.G. 1978. *Theoretical sensitivity: Advances in the methodology of grounded theory*. Mill Valley, CA: Sociology Press.

GLASER, B.G. 1998. *Doing grounded theory: Issues and discussions*. Mill Valley, CA: Sociology Press.

GLASER, B.G. 1992. *Basics of Grounded Theory analysis: Emergence vs. forcing*. Mill Valley, CA: Sociology Press.

GLASER, B.G. 1993. *Examples of Grounded Theory: A reader*. Mill Valley, CA: Sociology Press.

GLASER, B.G. 2002. Constructivist Grounded Theory? In *Forum Qualitative Sozialforschung / Forum: Qualitative Social Research*, vol. 3, no. 3. [Online]. Available: <http://www.qualitative-research.net/fqs/fqs-eng.htm> [Cited 15 March 2008].

GOEDE, R. 2005. *Framework for the explicit use of specific systems thinking methodologies in data driven decision support system development*. Pretoria: University of Pretoria (PhD Thesis, University of Pretoria). [Online]. Available: <http://upetd.up.ac.za/thesis/available/etd-05132005-080727/> [Cited 12 January 2008].

GOEDE, R. *Substantive area of research*. Personal e-mail (23 April 2008).

GOLDKUHL, G. 1999. The grounding of usable knowledge: An inquiry in the epistemology of action knowledge. In *Högskolor och Samhälle i Samverkan (HSS99)*, Linköping University, March 16-18 (CMTO Research Papers 1999:03).

GOLDKUHL, G. 2002. Anchoring scientific abstractions: Ontological and linguistic determination following socio-instrumental pragmatism. *Proceedings of the European*

Conference on Research Methods in Business and Management Studies (ECRM 2002), MCIL, Reading, UK, April 29-39.

GOMES, P. 2004. Software design retrieval using Bayesian networks and WordNet. *Proceedings of the 7th European Conference on Case Based Reasoning (ECCBR, 2004), Madrid, Spain, August 30 - September 2*, pp. 184-197. [Online]. Available: <http://eden.dei.uc.pt/~pgomes/> [Cited 14 August 2006].

GONZALEZ-PEREZ, C. & HENDERSON-SELLERS, B. 2006. An ontology for software development endeavors. In *Ontologies for Software Engineering and Software Technology*. Edited by Coral Calero, Francisco Ruiz and Mario Piattini. Berlin: Springer Verlag.

GREEN, R.M. 1998. Heuristic power as the test of theory: A response to Franscisca Cho. *The Journal of Religious Ethics*, vol. 26, no. 1 , pp. 175-184.

GREGOR, S. 2006. The nature of theory in Information Systems. *MIS Quarterly*, vol. 30, no. 3, pp. 612-642.

GREGOR, S. n.d. *The struggle towards an understanding of theory in Information Systems*. School of Business and Information Management, Australian National University. [Online]. Available: http://epress.anu.edu.au/info_systems/part-ch01.pdf [Cited 31 August 2006].

GRUBER, T. 2008. Ontology. In *Encyclopedia of Database Systems, edited by Ling Liu and M. Tamer Ozsu*. Berlin: Springer Verlag.

GRUBER, T.A. 1993. Transition approach to portable ontology specifications. *Knowledge Acquisition*, vol. 5, no. 2, pp.199-220.

GUARINO, N. 1998. Formal Ontology and Information Systems. *Proceedings of FOIS'98, Trento, Italy, June 6-8, 1998*, pp. 3-15. Amsterdam: IOS Press.

GUARINO, N., CARRARA, M. & GIARETTA, P. 1994. Formalising ontological commitments. *Proceedings of the 12th National Conference on Artificial Intelligence (AAAI-94), Seattle, WA, August 1-4*.

GUNTER, C.A., GUNTER, E.L., HILL, M.N.J. & ZAVE, P. 2000. Formal software engineering. *ACM SIGSOFT Software Engineering Notes*, January 2000, vol. 25, no. 1, pp. 54.

GUZZARDI, G. & HALPIN, T. 2008. Ontological foundations for conceptual modeling, Guest Editorial, *Applied Ontology*, vol. 3, no 1-2, pp. 1-12.

HACKING, I. 2002. *Historical Ontology*. London: Harvard University Press.

HALLER, A. & OREN, E. 2006. *A process ontology to represent semantics of different process and choreography meta models*. Gallway: National University of Ireland (DERI Technical Report).

HARRIS, M.A. & WEISTROFFER, H.R. 2009. A new look at the relationship between user involvement in systems development and system success. *Communications of the Association of Information Systems*, vol. 24, no. 42, pp. 739-756.

HARRIS, M.L., HEVNER, A.R., & COLLINS, R.W. 2009. Controls in flexible software development. *Communications of the Association of Information Systems*, vol. 24, no. 43, pp. 757-776.

HEINEMAN, G.T. 2000. A model for designing adaptable software components. *ACM SIGSOFT, Software Engineering Notes*, vol. 25, no. 1, pp. 55-56.

HENNING, E. 2007. *Finding your way in qualitative research*. Pretoria: Van Schaik Publishers.

HEVNER, A.R., MARCH, S.T., PARK, J. & RAM, S. 2004. Design Science in Information Systems research. *MIS Quarterly*, vol. 28, no. 1, pp. 75-105.

HIGHSMITH, J. 2004. *Agile project management: Creating innovative products*. Boston: Addison-Wesley.

HIRSCHHEIM, R., KLEIN, H.K. & LYYTINEN, K. 1995. *Information systems development and data modelling: Conceptual and philosophical foundations*. Cambridge: Cambridge University Press.

HOFFERER, P. 2007. Achieving business process model interoperability using meta-models and ontologies. In *Managing Worldwide Operations and Communications with Information Technology, Proceedings of 2007 Information Resources Management Association, International Conference, Vancouver, British Columbia, Canada, May 19-23*, edited by M. Khosrow-Pour. Hershey, PA: IGI Publishing, pp. 1620-1631.

HOHMANN, C. 2007. Emotional digitization. A reflexive examination from the view of the industry. *International Journal of Technology and Human Interaction*, vol. 3, no. 1, pp. 17-29.

HOLT, A.W. 1997. *Organized activity and its support by computer*. Dordrecht: Kluwer Academic Publishers.

HUNTER, G.M. 2000. "Excellent" systems analysis: A grounded theory approach to qualitative research. In *The Handbook of Information Systems Research*, edited by Whitman, M.E. & Woszczyński, A.B. Hershey, PA: Idea Group Publishing.

HUNTER, G.M. 2003. Qualitative research in information systems: An exploration of methods. In *The Handbook of Information Systems Research*, edited by Whitman, M.E. & Woszczyński, A.B. Hershey, PA: Idea Group Publishing.

IEEE. 1990. *IEEE Standard Glossary of Software, IEEE Standard 610.12-1990*. New York, NY: IEEE.

IIVARI, J. 1991. A paradigmatic analysis of contemporary schools of IS development. *European Journal of Information Systems*, vol. 1, no. 4, pp. 249-272.

IIVARI, J., HIRSCHHEIM, R. & KLEIN, H.K. 1998. A paradigmatic analysis contrasting information systems development approaches and methodologies. *Journal of Information Systems Research*, vol. 9, no. 2, pp. 164-193.

IN-PHARMA TECHNOLOGIST.COM. 2005. *Improved data dealing drives drug discovery*. [Online]. Available: <http://www.in-pharmatechnologist.com/news/> [Cited 30 March, 2005].

INTRONA, D. 1992. *Towards a theory of management information*. Pretoria: University of Pretoria (Unpublished DCom Dissertation).

INTRONA, L.D. 1997. *Management information and power: A narrative of the involved manager*. Basingstoke: Macmillan Press.

ISABELLA, C.A. 1990. Evolving interpretations as a change unfolds: How managers construe key organizational events. *The Academy of Management Journal*, vol. 33, no. 1, pp. 7-41.

- JACKSON, M.A. 1981. *A systems development method: Tools and notions for program construction*. Cambridge, UK: Cambridge University Press.
- JAYARATNA, N. 1990. Systems analysis: The need for a better understanding. *International Journal of Information Management*, vol. 10, no. 8, pp. 228-234.
- JOHN, R.R. 2003. When information come of age: Technologies of knowledge in the age of reason and revolution, 1700-1850. *William and Mary Quarterly*, vol. 60, no. 2, Reviews of Books.
- JURISICA, I., MYLOPOULOS, J. & YU, E. 1999. Using ontologies for knowledge management: An information systems perspective. *In Proceedings of the 62nd Annual Meeting of the American Society for Information Science (ASIS99), Washington, DC, Oct 31-Nov 4*, pp. 482-496. Medford, NJ: Information Today, Inc.
- KAPLAN, B. & MAXWELL, J.A. 1994. Qualitative research methods for evaluating computer information systems. In *Evaluating Health Care Information Systems: Methods and Applications* (J.G. Anderson, C.E. Aydin and S.J. Jay, eds.). Thousand Oaks, CA: Sage Publications, pp. 45-68.
- KAWALEK, P. & LEONARD, J. 1996. Evolutionary software development to support organizational and business change: A case study account. *Journal of Information Technology*, vol. 11, no. 3, pp. 185-198.
- KEEN, P.G.W. 1988. Roles and skills base for the IS organization. In *Elam, J.J., Ginzberg, M.J., Keen, P.G.W. & Zmud, R.W. 1988. Transforming the IS Organization*. Washington, DC: ICIT Press, pp. 17-40.
- KIRLIDOG, M. & AYTOL, M. 2010. A longitudinal investigation about issues and problems of software engineering in Turkey. *Proceedings of the IBIMA 2010 Conference: Business Transformation through Innovation and Knowledge Management: An Academic Perspective, Istanbul, Turkey, June 23-24*, pp. 66-76.
- KLEIN, H.K. & MYERS, M.D. 1999. A set of principles for conducting and evaluating interpretive field studies in Information Systems. *MIS Quarterly*, vol. 23, no. 1, pp. 67-94.

KROEZE, J.H. 2009. *Information Systems and the humanities: a symbiotic relationship?* Vanderbijlpark: North-West University, Vaal Triangle Campus. (Vaal Triangle Occasional Papers: Inaugural lecture 5/2009.)

KROEZE, J.H., LOTRIET, H., MAVETERA, N., POSTMA, D., PFAFF, M., SEWCHURRAN, K. & TOPI, H. 2010. Humanities-enriched Information Systems: Panel discussion (abstract). In *Proceedings of the 18th European Conference on Information Systems (ECIS), IT to Empower, University of Pretoria, June 6-10.*

KROEZE, J.H., LOTRIET, H.H., MAVETERA, N., PFAFF, M.S., POSTMA, D.J.V.R., SEWCHURRAN, K. & TOPI, H. 2011. ECIS 2010 panel report: Humanities-enriched Information Systems. *Communications of the Association for Information Systems Journal (CAIS)*, vol. 28, no. 1. Available at: <http://aisel.aisnet.org/cais/vol28/iss1/24> [Cited 27 October 2011].

KRUEGER, C.W. n.d. *Introduction to software product lines*. [Online]. Available: <http://www.softwareproductlines.com/introduction/introduction.html> [Cited 17 June 2008].

LATOUR, B. 1999. *Pandora's hope: Essays on the reality of science studies*. Cambridge, MA: Harvard University Press.

LEEDY, P.D. & ORMROD, J.E. 2005. *Practical research: Planning and design*. 8th ed. Upper Saddle River, NJ: Pearson Education International.

LEHMAN, M.M. 1991. Software engineering, the software process and their support. *Software Environments and Factories (special issue), IEEE Software Engineering Journal*, vol. 6, no. 5, pp. 243-258.

LEHMAN, M.M. 1994. Feedback in the software evolution process. Keynote address. In *CSR Eleventh Annual Workshop on Software Evolution: Models and Metrics*. Dublin, September 7-9, 1994. Also in: *Workshop Proceedings, Software Maintenance (special issue), Information and Software Technology*, vol. 38, no. 11, pp. 681-686.

LEHTOLA, L., KANPPANEN, M., VAHINIITTY, J. & KOMSSI, M. 2009. Linking business and requirements engineering: Is solution planning a missing activity in software product companies? *Requirements Engineering Journal*, vol. 14, no. 2, pp. 113-128.

- LEMMENS, R. 2006. *Semantic interoperability of distributed geo-services*. Delft: TU Delft (PhD Dissertation, Delft University of Technology). [Online]. Available: <http://www.ncg.knaw.nl/publicaties/geodesy/pdf/63lemmens.pdf> [Cited 17 June 2008].
- LIU, L. & YU, E. 2004. Designing information systems in social context: A goal and scenario modelling approach. *Journal of Information Systems*, vol. 29, no. 2, pp.187-203.
- LOCKWOOD, D. 1964. Some remarks on 'The Social System'. In Demereth, N.J. & Peterson, R.A. 1967. *System, change, and conflict: A reader on contemporary sociological theory and the debate over functionalism*. New York: Free Press.
- LOFLIN, L. n.d. *Romanticism notes*. [Online]. Available: http://www.sullivan-county.com/nf0/nov_2000/romanticism.htm [Cited 4 September 2006].
- MALINOWSKI, B. 1923. The problem of meaning in primitive languages. In *The Meaning of Meaning*, edited by C.K. Ogden et al. London: Routledge and Kegan Paul.
- MALUF, D. & WIEDERHOLD, G. 1997. Abstraction of representation for interoperation. In *Foundation of Intelligent Systems, Lecture Notes in Computer Science, Lecture Notes in AI*, vol. 1525, pp. 441-455. Berlin: Springer.
- MARCH, S.T. & SMITH, G.F. 1995. Design and natural science research on information technology. *Decision Support Systems*, vol. 15, no. 4, pp. 251-266.
- MARTIN, P.Y. & TURNER, B.A. 1986. Grounded Theory and organizational research. *The Journal of Applied Behavioral Science*, vol. 22, no. 2, pp. 141-157.
- MATAVIRE, R. & BROWN, I. 2008. Investigating the use of 'Grounded Theory' in Information Systems research. In *SAICSIT '08: Proceedings of the 2008 annual research conference of the South African Institute of Computer Scientists and Information Technologists on IT research in developing countries: Riding the wave of technology, Wilderness, October 6-8*, pp. 139-147.
- MAVETERA, N. & KADYAMATIMBA, A. 2003. A comprehensive agent mediated e-market framework. In *ACM International Conference Proceeding Series, Proceedings of the 5th international conference on electronic commerce, Pittsburgh, Pennsylvania*, vol. 50, pp. 158-164.

- MAVETERA, N. & KROEZE, J.H. 2008. Practical issues in Grounded Theory Method research. *In the proceedings of the M & D Graduate Symposium, preceding SAICSIT 2008, Wildernis, October 6.*
- MAVETERA, N. & KROEZE, J.H. 2009a. Issues affecting software development: A South African software practitioners' viewpoint. *Communications of the IBIMA*, vol. 9, no. 2. [Online]. Available: <http://www.IBIMA.org> [Cited 25 April 2009].
- MAVETERA, N. & KROEZE, J.H. 2009b. Issues affecting software development: A South African software practitioners' viewpoint. *In Proceedings of the International Business Information Management Association Conference (IBIMA), Cairo, Egypt, January 4-6.*
- MAVETERA, N. & KROEZE, J.H. 2009c. Practical considerations in Grounded Theory Method research. *Sprouts: Working Papers on Information Systems*, vol. 9, no. 32. [Online]. Available: <http://sprouts.aisnet.org/9-32> [Cited 5 February 2010].
- MAVETERA, N. & KROEZE, J.H. 2009d. A grounding framework for developing adaptive software products. *In Proceedings of the 13th IBIMA Conference on Knowledge Management and Innovation in Advancing Economies, Marrakech, Morocco, November 9-10.*
- MAVETERA, N. & KROEZE, J.H. 2010a. Considerations in Grounded Theory research method: A reflection on the lessons learned. *In Proceedings of the 14th International Business Information Management Association Conference (14th IBIMA), Business Transformation through Innovation and Knowledge Management: An Academic Perspective, Istanbul, Turkey, June 23-24*, edited by Khalid S. Soliman, pp. 1454-1469.
- MAVETERA, N. & KROEZE, J.H. 2010b. An ontology-driven software development framework. *In Proceedings of the 14th International Business Information Management Association Conference (14th IBIMA), Business Transformation Through Innovation and Knowledge Management: An Academic Perspective, Istanbul, Turkey, June 23-24*, edited by Khalid S. Soliman, pp. 1713-1724.
- MAVETERA, N. & KROEZE, J.H. 2010c. Guiding principles for developing adaptive software products. *Communications of the IBIMA*, vol. 2010, Article ID 340296.

MAVETERA, N. 2004a. The use of ontologies in designing agents for e-markets: From a mechanist to a romantic approach. In *Proceedings of the International Business Information Management Conference (IBIMA '04), Amman, Jordan, July 4-6*.

MAVETERA, N. 2004b. The philosophy of ontologies: A new information systems development paradigm. In *Proceedings of the International Science and Technology Conference (ISTC'04), Vanderbijlpark, December 1-2*.

MAVETERA, N. 2007. A comprehensive ontology-driven software development architecture: A holistic approach to developing romantic software products. In *Managing Worldwide Operations and Communications with Information Technology, Proceedings of 2007 Information Resources Management Association, International Conference, Vancouver, British Columbia, Canada, May 19-23*, edited by M. Khosrow-Pour. Hershey, PA: IGI Publishing.

MAVETERA, N. 2010. Philosophical groundings in social science research. In *Fundamentals of Scientific Writing and Publishing: Research and Publication Series, Volume 1*, Mafikeng: Platinum Press.

MAVETERA, N. 2011. Towards an ontology-driven software development approach. In *Proceedings of the 16th IBIMA Conference on e-Government, Innovation and Knowledge Management: A Global Competitive Advantage, Kuala Lumpur, Malaysia, June 25-July 2*, pp. 1050-1057.

MERTON, R.K. 1967. *On Theoretical Sociology: Five essays, new and old*. Free Press, New York.

MESO, P. & JAIN, R. 2006. Agile software development: Adaptive systems principles and best practices. *Information Systems Management*, vol. 23, no. 3, June 2006, pp. 19-30.

MINGERS, J. 2002. Can social systems be autopoietic? Assessing Luhmann's social theory. *Sociological Review*, vol. 50, no. 2, pp. 278-299.

MOLENAAR, M. 1998. *An introduction to the theory of spatial object modelling for GIS*. London: Taylor & Francis.

- MONOD, E. 2007. Editorial. Special issue on philosophy and epistemology: A Peter Pan Syndrome? *Information Systems Journal*, vol. 17, no. 2, pp. 133-141.
- MULLET, D. 1999. *The software crisis*. [Online]. Available: <http://www.unt.edu/benchmarks/archives/1999/july99/crisis.htm>. [Cited 9 June 2008].
- MYERS, M.D. 1997. Qualitative research in information systems. *MISQ Discovery*, archival version, vol. 21, no. 2, June 1997, pp. 241-242. [Online]. Available: http://www.misq.org/discovery/MISQD_isworld/ [Cited 9 April 2008].
- MYERS, M.D. 2003. Qualitative research in Information Systems. *Association for Information Systems*. [Online] Available: <http://www.qual.auckland.ac.nz/index.htm> [Cited 9 April 2008].
- NECHES, R., FIKES, R., FININ, T., GRUBER, T., PATIL, R., SENATOR, T. & SWARTOUT, W.R. 1991. Enabling technology for knowledge sharing. *AI Magazine*, vol. 12, no. 3, pp. 36-56.
- NGWENYAMA, O.K. & LEE, A.S. 1997. Communication richness in electronic mail: Critical social theory and the contextuality of meaning. *MIS Quarterly*, vol. 21, no. 2, June, pp. 145-167.
- OINAS-KUKKONEN, H. & HARJUMAA, M. 2009. Persuasive systems design: Key issues, process model, and system features. *Communications of the Association of Information Systems*, vol. 24, no. 28, pp. 485-500.
- OLIVIER, S.M. 2004. *Information Technology research: A practical guide for Computer Science and Informatics*. 2nd ed. Pretoria: Van Schaik.
- ORLIKOWSKI, W. J. & IACONO, S. 2001. Research commentary: Desperately seeking the “IT” in IT research: A call to theorizing the IT artefact. *Journal of Information Systems Research*, vol. 12, no. 2, pp. 121-134.
- ORLIKOWSKI, W.J. 1993. CASE tools as organizational change: Investigating incremental and radical changes in systems development. *Management Information Systems Quarterly*, 340, vol. 17, no. 3, pp. 309-340.

- PAGE, C. & MEYER, D. 2000. *Applied research design for business and management*. Australia: McGraw-Hill.
- PEFFERS, K., TUUNANEN, T., ROTHENBERGER, M.A. & CHALTERJEE, S. 2008. A Design Science research methodology for Information Systems research. *Journal of Management Information Systems*, vol. 24, no. 3, pp. 45-77.
- PRESSMAN, R.S. 2000. *Software engineering - A practitioner's approach*. 5th edition. Boston: Mc Graw-Hill Education.
- PRESSMAN, R.S. 2005. *Software engineering - A practitioner's approach*. 6th edition. Boston: McGraw-Hill.
- PURAO, S., BALDWIN, C., HEVNER, A., STOREY, V.C., PRIES-HEJE, J., SMITH, B. & ZHU, Y. 2008. The sciences of design: Observation on an emerging field. *Communications of the Association of Information Systems*, vol. 23, no. 28, pp. 523-546.
- RANDELL, B. n.d. *The 1968 NATO software engineering reports*. [Online]. Available: <http://homepages.cs.ncl.ac.uk/brian.randell/nato/natoreports/index.html> [Cited 2 November 2008].
- RAS, Z.W. & DARDZINSKY, A. 2004. Ontology based distributed autonomous knowledge systems. *International Journal of Information Systems*, vol. 29, no. 1, pp. 47-58.
- REETLEY, A. 2003. *A literature review on grounded theory*. Johannesburg: Rand Afrikaans University (Masters Dissertation).
- REPA, V. 2004. Methodology framework for information systems development. *CITSA Conference, Orlando, Florida, July 21-25*, pp. 2-5.
- ROODE, J.D. 1993. *Implications for teaching of a process based research framework for Information Systems*. Pretoria: Department of Informatics, University of Pretoria.
- ROODE, J.D. 2004. *Research methodologies and proposal (lecture)*. Pretoria: University of Pretoria (class notes).
- ROQUE, L., ALMEIDA, A. & FIGUEIREDO, A.D. 2003. Context engineering: An IS development approach. *Proceedings: Action in Language, Organisations and Information*

Systems, International Conference (ALOIS'2003), Linköping, Sweden, March 12-13, pp. 107-122.

ROSENKRANZ, C. & HOLTEN, R. 2007. Towards measuring the complexity of information systems: A language critique approach. *Proceedings of International Resources Management Association (IRMA), Vancouver, British Columbia, May 19-23*, pp. 57-60.

RUIZ, F. & HILERA, J.R. 2006. Using ontologies in software engineering and technology. In *Ontologies for Software Engineering and Software Technology*, edited by Coral Calero, Fransisco Ruiz and Mario Piattini. Berlin: Springer Verlag.

SARANTAKOS, S. 1997. *Social research*. 2nd ed. New York, NY: Palgrave.

SCHACH, S.R. 2005. *Object oriented and classical software engineering*. 6th ed. New York, NY: McGraw-Hill.

SEP. 2008. *Phenomenology*. Stanford Encyclopaedia of Philosophy (SEP). [Online]. Available: <http://palto.stanford.edu/entries/phenomenology/> [Cited 6 August 2009].

SHANKS, G. 1999. Semiotic approach to understanding representation in information systems. *Proceedings of the Information Systems Foundations Workshop on Ontology, Semiotics and Practice*. [Online]. Available: <http://www.comp.mq.edu.au/isf99/shanks.htm> [Cited 6 October 2005].

SHARMA, S. & INGLE, M. 2011. An ontology-driven information system. *International Journal of Computing Technology and Applications*, vol. 2, no. 1, pp. 147-154.

SHANKS, G., TANSLEY, E. & WEBER, R. 2003. Using ontology to validate conceptual models. *Communications of the ACM*, vol. 46, no. 10, pp. 85-89.

SHAWA, W. 2009. *An ontology for information systems development methodologies*. Potchefstroom: North West University (Master of Science Dissertation).

SHAWA, W., MAVETERA, N. & HUISMAN, M. 2009. Building language communities in organizational system development teams using ontologies. *Proceedings of the 13th IBIMA Conference on Knowledge Management and Innovation in Advancing Economies, Marrakech, Morocco, November 9-10*.

SIEGEMUND, K., THOMAS, E.J., ZHAO, U., PAN, J. & ASSMANN, U. 2011. Towards ontology-driven requirements engineering. *The 10th International Semantic Web Conference*, Bonn, October 23-27. [Online]. Available:

<http://iswc2011.semanticweb.org/fileadmin/iswc/papers/workshops/swese/4.pdf> [Cited 18 October 2011].

SIMONOV, M., GANGEMI, A. & SOROLDONI, M. 2004. Ontology-driven natural language access to legacy and web services in the insurance domain. *Journal of Business Information Systems*, pp. 1-10.

SMITH, H. 2000. The role of ontological engineering in B2B net markets. *CSC Europe*. [Online]. Available: <http://www.ontology.org/main/papers/csc-ont-eng.html> [Cited 21 May, 2004].

SOFFER, P., GOLANY, B., DORI, D. & WAND, Y. 2001. Modelling off-the shelf information systems requirements: An ontological approach. *Requirements Engineering Journal*, vol. 6, no. 3, pp. 183-199.

SOWA, F.J. 2000. *Ontology, metadata and semiotics*. In *Conceptual structures in logical, linguistic and computational issues*, edited by Ganter and Mineau. Berlin: Springer-Verlag, pp. 55-81 (*Lecture notes in AI #1867*).

SOWA, F.J. 2006. A dynamic theory of ontology. In *Formal Ontology in Information Systems*, edited by B. Bennett & C. Fellbaum. Amsterdam: IOS Press, Amsterdam [Online]. Available: <http://www.Jfsowa.com/pubs/dynonto.htm> [Cited 9 June 2008].

SOWA, F.J. 1976. Conceptual graphs for a data base interface. *IBM Journal of Research and Development*, vol. 20, no. 4, pp. 336-357.

SOWA, F.J. n.d. *Building, sharing and merging ontologies*. [Online]. Available: <http://www.JFsowa.com/ontology/ontoshar.htm> [Cited on 16 August 2006].

STAMPER, R. 1992. Signs, organisations, norms and information systems. In *Proceedings of the 3rd Australian Conference on Information Systems, University of Wollongong, Australia, October 5-8*.

- STOKES, D.R. & BIRD, J. 2008. Evolutionary robotics and creative constraints. In *Beyond the brain: Embodied, situated and distributed cognitions*, edited by Benoit Hardy-Vallée & Nicolas Payette. Newcastle: Cambridge Scholars Publishing, pp. 227-245.
- STRAUSS, A.L. & CORBIN, J. 1990. *Basics of qualitative research: Grounded Theory procedures and techniques*. London: Sage.
- STRUWIG, F.W. & STEAD, G.B. 2004. *Planning, designing and reporting research*. Cape Town: Pearson Education.
- STUDER, R., BENJAMINS, R. & FENSEL, D. 1998. Knowledge engineering: Principles and methods. *Data and Knowledge Engineering*, vol. 25, no. 1-2, pp. 161-197.
- SUCHMAN, L.A. 1987. *Plans and situated actions: The problem of human-machine communications*. Cambridge: Cambridge University Press.
- SUDDABY, R. 2006. From the editors: What Grounded Theory is not. *Academy of the Management Journal*, vol. 49, no. 4, pp. 633-642.
- SUGUMARAN, V. & STOREY, V.C. 2002. Ontologies for conceptual modeling: Their creation, use and management. *Journal of Data and Knowledge Engineering*, vol. 42, no. 2, pp. 251-271.
- SUGUMARAN, V. & STOREY, V.C. 2006. The role of domain ontologies in database design: An ontology management and conceptual modeling environment. *ACM Transactions on Database Systems (TODS)*, vol. 31, no 3.
- SUTTON, R.J. & STAW, B.M. 1995. What theory is not? *Administrative Science Quarterly*, ABI/Inform Global, September 1995, vol. 40, no. 3, pp. 371-384.
- SWARTOUT, B., PATIL, R., KNIGHT, K. & RUSS, T. 1997. Toward distributed use of large scale ontologies. *Ontological Engineering (AAAI-97)*, Spring Symposium Series, pp. 138-148.
- TANIAR, D. & RAHAYU, J.W. 2006. *Web semantics & ontology*. London: Idea Group Publishing.

- TARNAS, R. 1991. *The passion of the western mind: Understanding the ideas that have shaped our world view*. London: Pimlico.
- TRIM, P.R.J. & LEE, Y.I. 2004. A reflection on theory building and the development of management knowledge. *Journal of Management Decision*, vol. 42, no. 3-4, pp. 473-480.
- TROCHIM, W.M. n.d. *The research methods knowledge base*, 2nd edition. [Online]. Available: <http://www.socialresearchmethods.net/kb/> [Cited 7 April 2008].
- TRUEX, D., BASKERVILLE, R. & TRAVIS, J. 2000. A methodological systems development: The deferred meaning of systems development methods. *Journal of Accounting Management and Information Technology*, vol. 10, no. 1, pp. 53-79.
- TUCKER, W. 2009. It is not just a theory, it is a theory! *CAP Journal*, no. 5, pp. 9-10.
- TURBAN, E., MCLEAN, E. & WETHERBE, J. 2004. *Information Technology for Management: Transforming organization in the digital economy*, 4th ed. New York, NY: John Wiley & Sons, Inc.
- UNIVERSITY OF VICTORIA, n.d. *Criteria for assessing PhD thesis*. Faculty of Graduate Studies. [Online]. Available: <http://web.uvic.ca/gradstudies/pdf/phdcriteria.pdf> [Cited 12 May 2010].
- USCHOLD, M. & GRUNINGER, M. 2004. Ontology and semantics for seamless connectivity. *SIGMOD Record*, vol. 33, no. 4, December.
- VAN HEIJST, G., SCHEREIBER, A.T. & WIELINGA, B.J. 1997. Using explicit ontologies in KBS development. *International Journal of Human and Computer Studies*, vol. 46, no. 2/3, pp. 293-310.
- VAN NIEKERK, J.C. & ROODE, J.D. 2009. Glaserian and Straussian Grounded Theory: Similar or completely different? In *Proceedings of the 2009 Annual Research Conference of the South African Institute of Computer Scientists and Information Technologists, Vaal River, South Africa, October 13-14*, pp. 96-103.
- W3C. 2004. *Resource Description Framework (RDF): Concepts and abstract syntax*. [Online]. Available: <http://www.w3.org/TR/rdf-concepts/> [Cited 23 January 2009].

WALSHAM, G. 1993. *Interpreting information systems in organizations*. London, UK: John Wiley & Sons.

WAND, Y. & WEBER, R. 1990. Toward a theory of the deep structure of information systems. *Proceedings of the International Conference of Information Systems, Copenhagen*, pp. 61-71.

WAND, Y. & WEBER, R. 1993. On the ontological expressiveness of Information Systems analysis and design grammars. *Information Systems Journal*, vol. 3, no. 4, pp. 217-237.

WAND, Y. & WEBER, R. 2002. Research commentary: Information systems and conceptual modelling - a research agenda. *Journal of Information Systems Research*, vol. 13, no. 4, pp. 363-376.

WEBER, R. 2003. Conceptual modelling and ontology: Possibilities and pitfalls. Research paper review. *Journal of Database Management*, vol. 14, no. 3, pp. 1-20.

WHITTEN, J.L., BENTLEY, L.D. & DITTMAN, K.C. 2004. *Systems analysis and design methods*, 6th ed. Boston, MA: Irwin/McGraw-Hill.

WIKIPEDIA, n.d. *Ethnography*. [Online]. Available: <http://en.wikipedia.org/wiki/Ethnography> [Cited 20 August 2006].

WINTER, S. & TOMKO, M. 2006. Translating the web semantics of georeferences. In *Web Semantics and Ontology*, edited by D. Taniar & Rahayu, J. Hershey, PA: Idea Group Publishing, pp. 297-333.

WYSSUSEK, B. 2004. Ontology and ontologies in information systems analysis and design: A critique. In *Proceedings of the Tenth Americas Conference on Information Systems, New York, NY, August 5-8*, pp. 4303-4308.

YU, E. 1995. *Modelling strategic relationships for process engineering*. Toronto: University of Toronto (PhD thesis, Department of Computer Science). (Also Tech. Report DKBS-TR-94-6.)

YU, E.K.S. 1997. Towards modeling and reasoning support for early-phase requirements engineering. In *Proceedings of the 3rd IEEE Int. Symp. on Requirements Engineering (RE'97), Washington, DC, Jan. 6-8*.

ZÚÑIGA, G.L. 2001. Ontology: Its transformation from philosophy to Information Systems.
FOIS'01, 17-19 October 2001, Ogunquint, Maine, USA.

APPENDICES

Appendix A: Data Gathering Interview Questions

An Ontological Approach to Software Development: PhD Interview Research Questions.

Brief Introduction:

All information systems that are developed consist of three basic components, that is, database, user interface, and applications. Linking all these three is a common schema as shown in Figure A1 below (adapted from Sowa, n.d.).

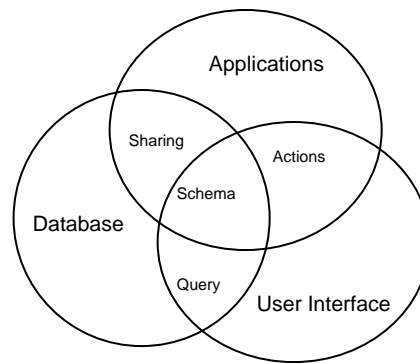


Figure A1: An Integrated Information System (Adapted from Sowa, n.d.)

The purpose of this research is to position ontology as an artefact that integrates all three, is part of each of the components and, at the same time, is a tool that can be used to develop evolvable software products with emergent properties. Such properties are the ones needed in fast-growing, interactive and evolving systems.

The research approach is a qualitative empirical investigation into the domain of software development. The researcher investigates practices around the development and use of software in information systems. An analysis of the technical design of the software and information systems is also part of the investigation.

Statement of the problem:

The problem that software developers continue to face is the lack of a method or tool that can augment current programming language technology and methodologies with semantics-based tools to enable construction of evolvable software products. Software kernels (Dittrich & Sestoft, 2005) and software product lines (Carnegie Mellon Software Engineering Institute, n.d.) have been used in industry, research, and academia to try to address the speed of reproducing and of the evolution of software products. Although they have gained widespread acceptance, they still leave a semantic gap that needs to be filled and incorporated in software products. An ontological approach to software development challenges most of the traditional software development approaches and borrows most of its characteristics from the agile methodologies family.

Aim of the Study

The aim of the study is to position ontology as an artefact that can be incorporated in software applications, either at run time or as a part of an information system, so as to increase semantic and context-awareness in information systems. This development will result in information systems that are more dynamic and usable with components that are reusable, shareable, visible and accessible to other systems and users as well.

Objectives of the Study

The objectives of the study can be summarized as the need to develop a general ontology framework that satisfies the aims of the study. The importance and validity of the framework will be justified using data gathered from the developers of information systems. More specifically the objectives of the study are:

- a. To develop or construct a tangible ontological theory within the field of software development that can be used to design and develop romantic information systems. The abstract ontology concept will be operationalized into a concrete form of knowledge that can be applied readily in software development (Agerfalk, 2004). Operationalization is the process of converting or moving abstractions to concrete artefacts that have practical applications in a social practice, such as an inter-organizational system.

- b. To develop a linguistic model that can be used by software agents to communicate in information systems.
- c. To position ontology as an artefact that is needed by software agents to enable them to develop the linguistic models needed for communication in information systems, both experientially and intuitively.
- d. To develop an ontology model-driven-approach to software development (El Baze, 2005). This approach will ensure that the resultant model moves from a purely abstract conceptualization of an ontology model to a truly operational model. This context-aware, purpose-specific actionable model could be deployed as a value-added service layer in software development processes.

Purpose:

The aim is to understand and explain the software development practices and designs from practitioners' view point. This aim is explained by – and based on – the historical and situational context of software development. The secondary aim of this research is to derive a new paradigm in systems development. The questions will be open and unstructured. In those situations that permit it, observation will be used to complement the data-gathering process.

Activities:

The researcher will investigate software development approaches, methods, techniques and tools currently used in software development. Considering the pervasiveness and ubiquity needed in current information systems, the activities will unravel the discrepancies between current development methods and the one proposed and strongly motivated by the literature study, that is: an ontology-driven development approach.

Section A: These few questions ask for demographic data

- i. For identification purposes only, can you tell me your name or the name of the company you represent?
- ii. How long have you been involved in software/system development?
- iii. What type of software products or information systems does your company specialize in?

iv. What is your role in the company?

Section B: These questions look at the different software/system development practices from the participants' view point.

These questions generally look at the paradigm aspects of software development.

Question 1:

There are several software development paradigms in widespread use today.

- a) Which paradigm do you advocate when developing software products in your organization?
- b) What is the main reason why you use this paradigm?

Question 2:

Some people say the difference between the structured methodology (use of process and data models and use of DFD's and ERD's among other techniques) and the object oriented paradigm (use of use cases, classes and activity diagrams with UML as a technique) is that the latter captures semantics of a system. What is your opinion of that?

The researcher needs to find how context and meaning in systems is captured during development time and at run time of software products.

'There is a reason why computers have not yet become fervent natural language speakers. (It's not a matter of processing power and never will be): we simply are not programming them correctly.'

(El Baze, 2005.)

Question 3:

What is your comment on this?

Question 4a:

It is common knowledge that information systems do not behave the same way as humans do. What do you suggest should be done to add this human-like behaviour in information systems artefacts?

Question 4b:

Information systems comprise three parts, the formal, informal and the technical part. Can you explain how the formal and informal part is captured during your software development processes?

Question 5:

Organizations as systems exhibit a lot of “unstated” assumptions that are reflected through the shared (common sense) knowledge of people familiar with the social businesses and technical contexts within which the proposed system will operate (Rosenkranz and Holten, 2007:58) the pragmatics. What can be done in software development to capture these assumptions?

Question 6:

Every organization runs information systems within an organizational culture and context. When developing software products, how can developers ensure that this organizational culture and context are captured and implemented in the software products?

The next group of questions looks at communication problems during software development.

Question 7:

- a) When communicating with clients, what communication tools and channels do you use in your organization?
- b) “It is easier for clients to understand the functionality of the software through the user interface sketches” (Dobing and Parsons, 2007:123). What is your view?

Question 8:

During software development, how can we improve the clients’ understanding of the functionality of the proposed software?

Question 9:

In which stages of software development would you involve the client?

Question 10:

How do you ensure that the understanding of the client of what the system should do is correct and maintained throughout the development process from analysis to maintenance stages?

Question 11:

Communication is a very important facet of software development process. Many people decry the lack of a communication language/tool that is understandable by both users (clients) and developers. What tools can be used to communicate between client-analyst and between analyst-programmer during software development?

Question 12: (*Check question 7 for similarity in answers*)

What type of clients do you have? Do they have knowledge of system development or software development tools?

Hint: What do you use to communicate with these clients?

The next group of questions looks at the ability of ontology-driven case tools to retain the system context gathered during the analysis to maintenance processes.

Question 13:

What do you think about a software development tool that integrates the software development tasks and data or information gathered from analysis through to implementation can improve on the software development process?

Question 14:

It is recommended that, when you develop a software development approach or methodology, one should also develop an accompanying development tool. The tool has considerable influence on how the approach or methodology will subsequently be used. What functions or

tasks would you want to see in a development tool that supports a development approach/methodology that allows adaptability and reusability in software products?

Question 15:

What is your understanding of adaptive software products?

Question 16:

How can you ensure the development of adaptive software products?

Question 17:

- a. What is your understanding of evolving software products or systems?
- b. How can you ensure the development of evolvable products?

This group of questions investigates, inter alia, issues affecting software development productivity. Ontology has the capability to improve software productivity, as will be discussed in the thesis.

Question 18:

How do you ensure requirements gathered during analysis and design specifications are reused?

Question 19:

As a software developer, you have developed software products for different organizations that are in the same industry. How do you reuse knowledge gained from one development process to the next project?

Techniques that mediate between a dialect conflict of context and meaning.

Question 20:

The success of any software development process depends on the ability to create a language community among stakeholders. This leads to the creation of shared knowledge that will later be captured in the resultant software product or system. The shared knowledge reflects itself through the concepts that will be used in a specific domain. How would you, as a developer,

mediate when working in the same industry, if the same concept is given different meanings? That is, there exists a dialect conflict of context and meaning (context versus meaning).

Software success metrics looking at the quality of software products.

Question 21:

How do you ensure that the software development process is?

- i. Within budget
- ii. On time
- iii. Easy to modify
- iv. Faulty free
- v. User is satisfied.

Question 22:

How do you manage scope creep (handle) at the same time ensuring all of the above?

The following questions look at design problems that can be handled using ontology. The technical nature of the ontology artefact requires questions to be very general and the research extrapolates the technical responses at the data analysis stage.

Question 23:

How do you capture semantics (meaning) of designs, specifications in your systems?

Question 24:

How do you ensure that systems are developed within a specific application (domain) context?

Question 25:

How can we improve the changing and communication of a software product design change during coding time?

Question 26:

Discuss some of the challenges that you have encountered pertaining to developing software products that are reusable.

Hints for Interviewer:

- *Design for reuse*
- *Cost of reusing the components*
- *Archiving and retrieval of reusable components*
- *Accessibility of the source code - if not home grown.*
- *Adaptability and evolvability of source code components*
- *Any other / maintainability*
- *Progress measure during software maintainability / development.*

End of Interview

Appendix B: Publications Generated from the Research Process

- MAVETERA, N. 2011. Towards an Ontology-Driven Software development Approach: In *Proceedings of the 16th IBIMA Conference on e-Government, Innovation and Knowledge Management: A Global Competitive Advantage*, Kuala Lumpur, Malaysia, June 25-July 2, pp.1050-1057.
- Mavetera, N. 2011. The use of GTM in Qualitative Research. In *Innovative Teaching, learning and Research Methods in Higher Education, Book Chapter*, L. Makondo & MA. Mokoena Eds., And Cork Publishers, 2011. To Appear.
- Kroeze, J.H., Lotriet, H.H., Mavetera, N., Pfaff, M.S., Postma, D.J.V.R., Sewchurran, K. & Topi, H. 2011. ECIS 2010 Panel Report: Humanities-Enriched Information Systems. *Communications of the Association for Information Systems Journal (CAIS)*, vol. 28, no. 1, article 24. Available at: <http://aisel.aisnet.org/cais/vol28/iss1/24>.

- d. MAVETERA, N. & KROEZE, J.H. 2010a. Considerations in Grounded Theory Research Method: A Reflection on the Lessons Learned. In *Proceedings of the 14th International Business Information Management Association Conference (14th IBIMA), Business Transformation through Innovation and Knowledge Management: An Academic Perspective, Istanbul, Turkey, June 23-24*, edited by Khalid S. Soliman. pp. 1454 - 1469.
- e. MAVETERA, N. & KROEZE, J.H. 2010b .An Ontology-Driven Software Development Framework. In *Proceedings of the 14th International Business Information Management Association Conference (14th IBIMA), Business Transformation through Innovation and Knowledge Management: An Academic Perspective, Istanbul, Turkey, June 23-24*, edited by Khalid S. Soliman. . pp. 1713 - 1724.
- f. MAVETERA, N. & KROEZE, J.H. 2010c. Guiding Principles for Developing Adaptive Software Products. *Communications of the IBIMA*, vol. 2010, Article ID 340296.
- g. KROEZE, J.H.; LOTRIET, H.; MAVETERA, N.; POSTHMA, D.; PFAFF, M.; SEWCHURRAN, K. & TOPI, H. 2010. Humanities-enriched Information Systems; panel discussion paper In *Proceedings of the 18th European Conference on Information Systems (ECIS), IT to Empower, University of Pretoria, June 6-10 2010*.
- h. MAVETERA, N. 2010. Philosophical Groundings in Social Science Research. In *Fundamentals of Scientific Writing and Publishing: Research and Publication Series Volume 1*, Mafikeng: The Platinum Press.
- i. SHAWA, W; MAVETERA, N. & HUISMAN, M. 2009. Building Language Communities in Organizational System Development Teams Using Ontologies. *Proceedings of the 13th IBIMA Conference on Knowledge Management and Innovation in Advancing Economies, Marrakech, Morocco, November 9-10*.
- j. MAVETERA, N. & KROEZE, J.H. 2009a. Issues Affecting Software Development: A South African Software Practitioners' Viewpoint. *Communications of the IBIMA*, vol. 9, no. 2, [Online]. Available: <http://www.IBIMA.org>. [Cited 25 April 2009].

- k. MAVETERA, N. & KROEZE, J.H. 2009b. Issues Affecting Software Development: A South African Software Practitioners' Viewpoint. *In Proceedings of the International Business Information Management Association Conference (IBIMA)*, Cairo, Egypt, January 4-6.
- l. MAVETERA, N. & KROEZE, J.H. 2009c. Practical considerations in Grounded Theory Method Research. *Sprouts: Working Papers on Information Systems*, vol. 9, no. 32. [Online]. Available: <http://sprouts.aisnet.org/9-32>.
- m. MAVETERA, N. & KROEZE, J.H. 2009d. A Grounding Framework for Developing Adaptive Software Products. *In Proceedings of the 13th IBIMA Conference on Knowledge Management and Innovation in Advancing Economies, Marrakech, Morocco, November 9-10*.
- n. Mavetera, N. 2009. The Non Linearity Nature of Choosing a Research Method, *In Proceedings of the Postgraduate Supervision Conference, Stellenbosch University, April 27-30*.
- o. MAVETERA, N. & KROEZE, J. H. 2008. Practical issues in Grounded Theory Method Research. *In Proceedings of the M & D Graduate Symposium, preceding SAICSIT 2008, Wilderness, October 6* Shawa, W. C.; Mavetera, N. & Huisman, M. 2008. An Ontology for Information Systems Development Methodologies. *In Proceedings of the M & D Graduate Symposium, preceding SAICSIT 2008, Wilderness, October 6*.
- p. MAVETERA, N. 2004a. The Use of Ontologies in Designing Agents for E-Markets: From a Mechanist to a Romantic Approach. *In Proceedings of the International Business Information Management Conference (IBIM '04), Amman, Jordan, July 4-6*.
- q. MAVETERA, N. 2004b. The Philosophy of Ontologies: A new Information Systems Development Paradigm. *In Proceedings of the International Science and Technology Conference (ISTC'04), Vanderbijlpark, December 1-2* MAVETERA, N. 2007. A Comprehensive Ontology-Driven Software Development Architecture: A Holistic Approach to Developing Romantic Software Products, *In Managing Worldwide Operations and Communications with Information Technology (Proceedings of 2007*

*Information Resources Management Association, International Conference,
Vancouver, British Columbia, Canada, May 19-23,*

Appendix C: Incidents for Proposition PA

In these appendices, the italicized phrases indicate a word or phrase that supports the hypothesis under discussion.

P 2: The *realistic world view* says that everything is agreeable. You and I can decide this is a mouse and this is the function of the mouse. That is *the realistic world view* and that is where the problem with the engineers comes in. They follow a *realistic world view*. They think they agree on the purpose of this thing.

P 2: And if we follow a *relativistic world view*, you might have an opinion on this device and I can have a different opinion on this device. And if we then get down to a discussion, we might design a much better mouse than this - if we sit down and think “what is the real purpose”.

P 3: We shouldn't forget paradigms where they focus on people, *people-orientated paradigms*. We say, in the end, all the work we do, the systems we develop, the products we develop, is there to benefit people. So you should really be in contact with your users and so on. So that seems more *it's a social aspect* - your systems development.

Appendix D: Incidents for Proposition PB

P 1: So I think the best way to actually represent that information will be to come up with some - *not a framework - but maybe language* of some sort *that has the ability to actually express those concepts and to relate to technical issues that way*. I am tending to think in terms of the *approaches to development*.

P 1: Traditional approach, where you have a *systems analyst*, you have a *programmer*, and then you have got *the business guy, the user*. I call that the traditional approach. In the approach that I've studied in detail and also used, you'll find that *the gap between the analyst and the programmer is actually closed by coming up with somebody who's called a developer*, who actually *elicits the requirements directly from the user*. And that *user becomes part of the development team* which means that those details - those metaphors and the language that is used by the team becomes familiar to that user.

P 2: I'm very much *against the traditional paradigm* because I believe it leads to a case where you have *a system that conforms to the requirements and not to the user's needs*.

P 1: In *agile methodologies* - you'd actually *have the client from the beginning up to the end*.

P 2: But the best way is to *have a user onboard*, such as all these *agile methodologies*,

P 2: Object-oriented programming to me works well in the technical applications. Perhaps it's not yet proven enough for me in today's software development, except perhaps for requirements analysis.

P 2: *Object-oriented technique does allow more for people to capture all their requirements, to capture non-technical things*.

P 1: Soft issues- I would say [that] if an individual wants to work in that area, they'll have to look into a lot of other areas that may not necessarily be in IT but others are IT. For example going into psychology, going into organizational

behaviour. Those are areas that are very, very interesting and there are books that people have written that bring the softer issues of IT and the technical issues together.

P 1: Language Requirement-Because the *language is the only solid thing* or it's the only *output that you get from a human being that lets you know what is happening in their brain*. And if you want to deal with the informal part of the system, you are really looking at the softer issues, *the human issues of software development*. The *behaviour of human beings and their languages would be the centre that people have to deal with in order to understand*.

P 3: Soft Systems Methodology - The other paradigm would be - I would call holistic paradigm where they focus on the system - the systems thinking.

P 3: I would advocate a *contingency approach*, meaning I would first of all go and look at the *type of project with the type of system* that I would have to develop, look at the *characteristics of this system* and then I would decide. Say for example it's a life critical system; you have to develop something for people in the hospitals or so. Then I would go for the *structured approach*, mainly *because it's very, very focused*; you know exactly what to do next. I would *adjust the structured approach a bit to make it iterative*, so you can *go back to previous stages* if you think there was a mistake there and because of the documentation.

P 3: In my research I saw that *those two are the mostly used approaches in the industry*. You find that there's a *movement towards rapid application development* but the *structured and the object-orientated approach are the most used approaches*.

P 1: In Extreme Programming you have a practice that is called Pair Programming. In Pair Programming you put two people - two programmers - on one machine and that way, *that knowledge which cannot be written down but which is in those developers' minds, can actually be exchanged as they do the development*.

P 1: But it should take a much longer study where you're trying to marry all

those different fields because, if you are going to do such a task, you need to *know how people behave* and that's a different field from IT. So *the behaviour of human beings in different environments* - how do they do that.

P 1: Now that they [computers] can solve a good number of those problems, we now expect them to help us in thinking as well, so that we have a thinking machine that can solve some of your thinking problems. In that area, that's where I would say we are lacking. *We don't have the skills at the moment to actually come up with systems like that.* The reason being that *the human brain itself and the behaviour of the human being, those two issues are still grey* even to those who are specialists in those areas.

Appendix E: Incidents for Proposition PC

P 1: So one approach that should be used to actually capture that kind of [tacit] knowledge would be to allow in the systems development process *a [discussion] forum or some place within the process where that knowledge can be exchanged within the individuals who are involved in the development team.*

P 1: So these *whiteboards* would be - *that's where people would just write those terms that they are used to, that they normally use.* And then the *flyers* would actually - *when people have a meeting they talk and so forth - then you stick those along the corridors in that organization.*

P 1: User Involvement - You don't want the systems analyst to meet some guys there, get some documents signed and then go and shift that information into the programme, but *you want these three levels [the analyst, the programmer and the user] to actually sit together.* *The user must be the centre of all those things.*

P 1: Quality Measure - So the *client [user] must be there throughout and that is basically for quality checks purposes, to ensure that the kind of requirements that the user has are actually met.*

P 2: *Involving your users in all the stages, so that we don't spend a lot of time on things*

that's not important.

P 1: So with user stories, *you'll ask the user to give you a different story of what functionality they would like to be implemented*. It's not as detailed as a *use case* but you give them some *6 x 6 inch cards* and ask them to write a little functionality that they want implemented. So you're breaking down the requirements into those small user stories.

P 3: What they [developers] do, *they get the requirements on user cards*. You know the *user stories*? Normally they're the small cards like [...] this size - and then *the users would write their requirements here of the system*. And what they do, in the end *they collect all the cards and they put a price tag on it*. The price in terms of the *time* it would take and the *physical resources* they would need and the *money*. And then they tell the users "what would you like us to do in the following three weeks".

P2: *If you show them these prototypes or these interface sketches*, like you call it, that *they would understand it much better*. To them *the system is the interface*.

P 1: *Proof-of-concept* is more practical, where you're coming up with *a demo of the system*, how the system is going to work. That's basically what really impressed them because they could *now start contributing* and saying "okay, at this point we want this" or "at this point, no, we don't deliver this kind of reports.

P 2: A *prototype* tells you sometimes that - it's empty behind that screen and *the proof-of-concept has some functionality* and you should have a bit of functionality.

P 2: To do a *very good verification after the requirements has been gathered*, can be done, once again, with the *proof-of-concept*.

P 1: When you *look at an object*, it gives a *better understanding of the system...* but I *wouldn't call it semantics*. I think *it's lighter than semantics*. It doesn't give the details that semantics would give. It, however, *leaves you at a general level* and that is an *advantage of object-oriented programming*. It gives a general class you can actually inherit and start reusing in the system. So I would say *there's a level of understanding that has been increased, as opposed to structured programming* but they still need to go deeper which would now become the semantic level.

P 1: [Language Limitations] For example, *when somebody expresses their idea, the*

limitations of that expression or that representation, are within the language that they are using.

P 3: [Communication problem between the conceptual and physical design.]

I would say maybe some of the environments can handle that but sometimes it's difficult. It's up to a certain stage. Once you've generated your physical database, it's difficult. So I agree, it could help but I would first of all really test the system or - no, not the system - the environment to see that it's done, it can handle it.

P 3: [Language Requirement] *Plain common language*; not technical jargon. Sometimes we don't even - we are not familiar with our own language, the terms we use, the slang that we use. So we should really, *really use just plain common language at the level that the user can understand.*

P 3: I think sometimes - you know, nowadays people are very interested in business analysts or business analytics or whatever they call it, where you have business people talking to your users. But I would go further, I wouldn't just have a business person there, I would like a technical person with business knowledge to do this, to do the communication

P 3: [Communication Problem] And I - really I would argue that it's because the client can't define the requirements correctly because we don't understand what exactly this user wants.

P 1: So that is exactly the demise that we have as software developers, that if we try to go into the details of the softer side - the non-formal parts of the system - we find that we have limitations in terms of the language. Because *whatever language you are using, when I try to use now the computer language to express that, it might not represent what you are talking about.* So it is really the issues of language that should be dealt with and so forth.

P 1: But what usually happens is that when you get to an organization, you need to *understand their language, the way they talk about things, whether they are technical or non-technical* but for the purposes of your requirements. The way the information is represented will sometimes differ from other organizations even in the same field.

P 2: The other way is to *incorporate the clients right through the process and not only in*

the requirements phase - but to talk to clients right through.

P 2: You can use UML quite well. But *one should not stick to the rigidity of such techniques or the prescriptiveness*; one should *be able to add your own things*, like to be flexible. *We don't have enough space on those diagrams to write down what its purpose originally was, and that is to capture non-technical information.*

P 2: That's why *ERDs are so difficult and not a very good tool to use, except in the physical design*. It's *not a requirements validation tool* for end-users.

P 2: [Star schemas] are the *basic design in data warehousing*. They replace the ERD. They replace the ERD but they are designed in such a way that I can teach you within three seconds how a star schema works but I won't even attempt to teach anybody an ERD.

P 1: [schemas] So now what I'm trying to say is that you need to come up with some schemas because that language is full of these schemas that you're used to, for example, you're trying to come up with a system for the library. In order for you to represent the requirements there, you come up with those schemas that will give you, for example, the - you can have schema called "loaning a book" or something like that.

P 3: The *user interface sketches* that you talk about could be used [to communicate system understanding with the users]. If you show *users* these prototypes or *these interface sketches*, they would understand it much better. To them the *system is the interface*.

Appendix F: Incidents for Proposition PD

P 1: *Iterative development*", where you develop your system in small iterations, and at the end of each iteration, you actually deliver a part of the system that is actually working.

P 2: Incremental development helps a lot *to do delivery in smaller pieces*, and then if you go wrong you don't have to backtrack the whole system. So *iteration and an incremental development* help a lot.

P 1: So an adaptive application would be an application that is highly maintainable, where you can get into the different modules and components of the system, and you do maintenance, *put new additions into that system without overburdening the system*, without

getting what is called “spaghetti cord” where the cord is no longer maintainable.

P 1: [Adaptive Products] should be products that, when you get into the maintenance phase, *they can easily be changed*. Software maintenance, basically it’s not about worn-out parts that you are replacing, it’s mainly about *bringing in the new changes that are relevant to the system in its environment*.

P 2: [Adaptive software products] means that the *product can change according to business needs* and it *should not take too long to make those changes*. And business needs can be requirements or it can be financial constraints or time constraints. If they run out of time you should be able to give them something at least usable. But mainly it should *be able to adapt to changes in business and business strategy within reasonable boundaries*.

P 3: [Adaptive software products] would be systems that can *change quickly according to changing environments and requirements*

P 1: [Software evolution] So his (this???) concept is that if you go [into] electrical engineering - a field of electrical engineering called “*control engineering*” - you’ll find that, in order to stabilise a system, they put all systems *in black boxes*, especially the processes - industrial processes. They’ll put it in a black box and say “a system has an input, and output”. And then, in order to stabilise that system, you are saying the output must be taped as either a part of the output or the entire output. You’ll tape it and fit it back into the input, so there must be a mixer there - some function which you represent by a summation sign to say “we are summing the input and part of the feedback”. But in so doing you are regulating the output so that the output does not spiral exponentially. Right, and that concept of spiralling exponentially you’ll find it - let’s say in speakers - if you put a microphone next to the speaker, then you have negative feedback. So you are trying to control the feedback there.

P 1: [Software Evolution] So in software evolution now the Lehman concept actually says that the moment you deploy the system and the user starts using the system, they’ll tell you some things that they don’t like about the system and whatever - things that need to be changed about the system. So that should be considered as feedback from the system that you have delivered. So *take that feedback and feed it together with the input, to control the way your system should actually work*. So that’s the concept. It’s trying to use that control

engineering approach to actually regulate the way we develop our systems.

P 1: Because *software development is not like other IT fields, like traditional engineering* and so forth. You find that you can't come up with blueprints and put them there and say [that] people are going to develop according to this, and they follow that because basically *things are based on the human brain*, it's more like an art.

but I would say the object orientation really could be more meaningful.

P 3: [Development Problem] And I - really I would argue that it's because the (client can't??) define the requirements correctly because we don't understand what exactly what this user wants. So if we put more effort into requirements gathering, I think you will see that problem - I wouldn't say it'll be solved - but much better. I think *we are so much in haste to get the product delivered, that we don't even bother to look at the requirements.*

P 3: And another thing I would also be careful though is, if companies buy into one of those software engineering environments - say for example Oracle Designer - that software engineering environment is essentially a companion to a methodology. For example, Oracle Designer, the companion methodology I would say is like information engineering. They call it something else but in essence it's information engineering. And if that methodology is not understood and in place in the company, and you use this software engineering environment, it's difficult. They find it difficult to do that. But if they know the methodology and they buy into this software engineering environment, then it becomes (inaudible - speaking softly). I call a software engineering environment, a methodology companion, those should go together. You can't have the one without the other one.

P 3: [Language Community] And there should be users at our side when we develop these requirements, and we should give them examples.

P 3: [Developer Understanding of Business Model] So, once again - so you walk into a company and you have to find out what these assumptions are. That's the way. Once again, a deeper of knowledge of business and communication. I think that's a big problem for consultants, if they go into the business and they just have to do consultancy (inaudible - noise) straight answer in my opinion would be communication and, hang on a bit, just get to know the business. And after you've done that for a period, you can start with your work.

P 3: [Problems with case tools] So it could help. In my experience through my research, I discovered that sometimes the analysis takes longer than what the people or the developers were used to and then they get discouraged. They should just keep with

the process, although it takes longer.

P 3: [Scope creep] Is spent on 20 percent of the requirement. Are they willing to do that? Are they willing to spend 80 percent of the time? If they cut that 20 percent of the requirement, they could save 80 percent of the time you can go onto a next project. So just discuss it with them. Ask them what their philosophy is, I would say, because some people really want that product and they want it in the way they visualise it and they want to add stuff and so on. And if they are willing to pay for that, fine with me, then I will add it because in the end of the day, it's [them] that are paying for the product, not me.

But other companies are very, very, very [results] driven and they want you to produce quick results and then I would show them. Think about it, if you are in your class also, the amount of time you spend with students in your office. It could be one student that takes up all your time in the office. It's the same with this principle. There's a small amount of requirement and it takes 80 percent of your time.

Appendix H: Full list of interview audio recordings (on CD)

Appendix I: Full list of interview transcripts (on CD)

Appendix J: Full list of incidents and codes for all the interviews (on CD)

Appendix K: Network diagram (on CD)

Appendix L: Full List of codes and code families for all the interviews (on CD)

Appendix M: Full list of incidents and memos for all the interviews (on CD)

Appendix N: Definition of Concepts Used in the Thesis

IT artefact: Orlikowski and Iacono (2001:1) refer to this as “bundles of material and cultural properties packaged in some socially recognizable form such as hardware and /or software”. On the other hand, Hevner *et al.*, (2004:77) regard it as “constructs (vocabulary and symbols), models (abstractions and representations), methods (algorithms and practices) and instantiations (implemented and prototype systems)”. In this research, however, the IT artefact takes into consideration the culturally and socially recognizable organizational

attributes, which at the same time, should be bundled into abstract constructs, which exhibit and inform people's practices.

Approach: Checkland (1999) refers to an approach as a way of going about tackling a problem. However this approach may not be very prescriptive on the method to be used when tackling such a problem. (See *Section 4.7 for more detail.*)

Methodology: This can be regarded as a formalized approach or as a combination of steps and deliverables that are needed for the development of a software product (Dennis *et al.*, 2001). This, however, comes at a lower level of the hierarchy than the approach discussed above. Benson and Standing (2005:203) somewhat philosophically define systems methodology as a “series of steps that are used in solving a problem to a general approach to problem solving”.

In contrast to an approach, a methodology as described includes the philosophical underpinning found in an approach, in which the “phases, procedures, rules, techniques, tools, documentation management and training for developers” are also listed (Benson & Standing, 2005:203). In research, however, a methodology can be conceived as a “model, which entails theoretical principles as well as a framework that provides guidelines about how research is done [...] in the context of a particular paradigm” (Sarantakos, 2004:32).

Model: Aßmann *et al.* (2006) describe a model as a representation, description and specification of a thing. The sole purpose of creating models is to portray reality faithfully. Although models are abstractions of such reality, any manipulations or queries made using the model should be reliable adaptations of the actual thing that they represent. A model can also be viewed as a pattern, plan, representation (especially in miniature) or as a description designed to show the main object or workings of an object, system or concept.

Information Systems: From an academic point of view, Roode (1993) defined an information system as ‘*an interdisciplinary field of scholarly inquiry, where information, information systems and the integration thereof with the organizations are studied in order to benefit the total system (technology, people, organizations and society)*’. In practice, Turban *et al.* (2004:18-19) distinguish between formal and informal information systems but the overarching characteristic of these systems is their ability to “*collect, process, store, analyze and disseminate information for a specific purpose*”.

Mechanistic Systems: These are systems that are very efficient at capturing and structuring data to enable and facilitate its interpretation. As Roode (1993) states, processing data and providing ‘information’ using technology is only the first step in a series of activities that lead to the provision of information as understanding. Mechanistic systems are based on the concept of explicit programming, which leaves the resultant system unable to provide its users with semantically rich information. The mechanistic notion assumes that the world is rational, deterministic and can be described sufficiently using rules and algorithms.

As Struwig and Stead (2004) state, the mechanistic world view conceives reality as existing and a given. Mechanistic software products are, therefore, developed using this notion of the mechanistic world view.

Romantic Systems: These are systems that possess a certain degree of humanistic behaviour. They are open and non-deterministic and unlike machines, do not subscribe to mechanistic ideas of representation, formalization, program, order, reason, stability or control. Their behaviour incorporates the importance of power struggles, changing practices and impending chaos. They borrow their definition from romanticism (Basden, 2001; John, 2003; Gregor, n.d. and Loflin, n.d.), which may imitate belief systems that depend on ‘irrationalism and feelings’.

According to Struwig and Stead (2004), the romantic world view posits reality as a social construction and believes in a shared reality among actors. Most importantly, what may be considered as knowledge depends heavily on the context, organizational politics and cultural factors as discussed in Chapter 4. Romantic software products should, therefore, be developed using this notion of romanticism. However this notion has practical limitations as will be discussed in this thesis.

Ontology: The ontology of a system can be described by defining a set of representational terms in the universe of discourse (Studer *et al.*, 1998; Swartout *et al.*, 1997; Mavetera, 2004a). It is an artefact that represents the acquisition, storage and dissemination of knowledge in information systems. In ontology, definitions associate the names of entities in the universe of discourse with human-readable text describing what the names mean and with formal axioms that constrain the interpretation and well-formed use of these terms.

Appendix G: Incidents for Proposition PE

P 2: [Requirements Repository] I don't think the end-user will benefit as much as the developer from doing it because you have one version of the truth of the requirement, you have one place - central repository - where you can find the requirements, you can set up checklists to see if you have fulfilled the requirements, the requirements are nearby but then the system must force the developers or the analysts to fill it in.

P 2: [Knowledge reuse] We call it experience. One thing one can say is that *involvement of the right people from the start*. If you've written a system and you - normally when it fails it is kind-of because that the right people was not involved from the company. That's my experience that the right people were not involved from the start and then that kind of experience I will take along and do a readiness assessment before I do anything else, and get the right people involved before I start working again on a new project.

P 2: [Scope creep] I think scope creep always happens but the extent of it depends on your - two things - actually mainly on the quality of your requirements. If your requirements are done well and *you involve the right people, you might have a situation where you don't get that much scope through it*. And then as - if you are using a *methodology where your end-user is involved, then scope should be their problem, not your problem* in a sense. But that's only the case if you have an end-user or a - I don't want to use the word end-user; I want to use a word sponsor user in data warehousing, *business sponsors* - somebody higher up in the organization. Somebody more important than the secretary that's going to type on the system; somebody that has power inside the organization. I want to have that person involved and he should prioritise all these additional scope requirements, not the programmers.

P 3: [Ambiguity on methodology that captures semantics] I think if you really, you do your ERs, your diagrams, and your dataflow diagrams, if you do that correctly and you explain it correctly to your user, I would say there could be semantics. Let me just think about it - that question. The object orientation, if you look at the use case diagrams, and you get the documentation of all the use cases, okay that could be more meaningful in a sense. There could be some semantics in there. Okay, I would say both

Framework: A framework is a conceptual structure that can be used to solve problems. An ontological framework, therefore, proposes ontology components that should be incorporated into software development practices.

Paradigm: According to Sarantakos (2004:31), a paradigm can be considered as a “set of propositions that explain how the world is perceived, it contains a world view, a way of breaking down the complexity of the real world”. It is a holistic aggregation or representation of the values, beliefs and techniques that are shared by a specific scientific community. The same paradigm dictates the types of problems and solutions to these problems that can be investigated and how they are investigated.

Theory: In this study, it will be regarded as a soft, fluid set of agreed thinking, reached by similar-minded actors in an application domain. This agreed thinking makes up a coherent body of knowledge that can guide an actor to execute a certain action in a social practice (Agerfalk, 2004). This is also supported by Gregor (2006), Tucker (2009) and Sutton and Staw (1995).

Appendix J: Full list of incidents and codes for all the interviews (on CD)

HU: An Ontological Approach to Software Development
File: [C:\Documents and Settings\NMavetera\My Docum...\An Ontological Approach to Software Development.hpr5]
Edited by: Super
Date/Time: 10/09/03 05:40:09 PM

Code: adaptive evolutionary approach.. {1-0}

P 5: [adaptive evolutionary approach..] (121:121) (Super)
Codes: [adaptive evolutionary approach..]

adaptive evolutionary approach,

Code: Adaptive software products. {5-0}

P 1: [It should be products that whe..] (115:115) (Super)
Codes: [Adaptive software products.]

It should be products that when you get into the maintenance phase, they can easily be changed. Software maintenance, basically it's not about worn-out parts that you are replacing, it's mainly about bringing in the new changes that are relevant to the system in its environment.

P 1: [So an adaptive application wou..] (115:115) (Super)
Codes: [Adaptive software products.]

So an adaptive application would be an application that is highly maintainable, where you can get into the different modules and components of the system, and you do maintenance, put new additions into that system without overburdening the system, without getting what is called "spaghetti cord" where the cord is no longer maintainable, it's growing beyond whatever and so forth.

P 2: [It means that the product can ..] (83:83) (Super)
Codes: [Adaptive software products.]

It means that the product can change according to business needs and it should not take too long to make those changes. And business needs can be requirements or it can be financial constraints or time constraints. If they run out of time you should be able to give them something at least usable. But mainly it should be able to adapt to changes in business and business strategy within reasonable boundaries.

P 3: [It would be systems that they ..] (168:168) (Super)
Codes: [Adaptive software products.]

It would be systems that they change quickly according to changing environments and requirements

P 8: [an application should be ... lik..] (164:164) (Super)
Codes: [Adaptive software products.]

Code: Adaptive Systems {2-0}

P 5: [heuristic systems that could l..] (79:79) (Super)

Codes: [Adaptive Systems] [heuristic systems]

heuristic systems that could learn from the way that a person uses the system, and sort-of adapt its response to become more and more inline with how that person uses it.

P 8: [It means we should leave room ..] (160:160) (Super)

Codes: [Adaptive Systems]

Code: Adaptive systems development approaches {3-0}

P 5: [In a crafted quality or agile ..] (119:119) (Super)

Codes: [Adaptive systems development approaches] [timebox approach]

In a crafted quality or agile environment, I certainly don't make promises about things like that, and I say "give me a timebox, let's set some goals and objectives for that timebox and we will do the best we can". And at the end of the timebox we will review what we have achieved. So I can certainly ensure that the budget is adhered to because the timebox gives me a certain budget. I can achieve on time because the timebox is limited, say to six weeks, two months, three months, whatever; but I can't guarantee that I what I build will be easy to modify, fault free, and that the user will be satisfied. That we will see - on reflection see to what extent we have achieved that. If the customer says "you've achieved enough to gain my confidence" then we get another timebox to improve those things.

P 8: [So, like I said, it's basicall..] (112:112) (Super)

Codes: [Adaptive systems development approaches] [Requirements of a new development approach] [Romantic Worldview]

P 8: [In comparison to an applicatio..] (162:162) (Super)

Codes: [Adaptive systems development approaches]

Code: Advantages of Agile Approach {2-0}

P 7: [I think for now Agile System D..] (41:41) (Super)

Codes: [Advantages of Agile Approach]

P 7: [I would say Agile System Devel..] (45:45) (Super)

Codes: [Advantages of Agile Approach]

Code: Agile Approach Concept {2-0}

P 5: [crafted quality". Now crafted ..] (113:113) (Super)

Codes: [Agile Approach Concept] [crafted quality"]

crafted quality". Now crafted quality is very much what most people call agile (inaudible - cross-talking) okay; and controlled quality is very much the kind-of conventional waterfall lifecycle approach.

P 9: [Agile method because everyone ..] (25:25) (Super)

Codes: [Agile Approach Concept]

Code: agile approach. {1-0}

P 5: [But in situations where the so..] (1:1) (Super)

Codes: [agile approach.]

But in situations where the solution can't be well-defined - maybe it's more of an experiential or research type of

initiative or where there's a tremendous amount of pressure to deliver results rapidly and therefore there isn't enough time to do thorough planning - then I would recommend more of an agile approach

Code: agile methodologies {4-2}

P 1: [agile methodologies] (25:25) (Super)

Codes: [agile methodologies]

agile methodologies

P 1: [in the agile methodologies - y..] (107:107) (Super)

Codes: [agile methodologies]

in the agile methodologies - you'd actually have the client from the beginning up to the end.

P 2: [But the best way is to have a ..] (63:63) (Super)

Codes: [agile methodologies]

But the best way is to have a user onboard, such as all these agile methodologies,

P 2: [extreme programming and new de..] (67:67) (Super)

Codes: [agile methodologies]

Memos: [User understanding]

extreme programming and new development,

Code: Agile System Development Lifec.. {1-0}

P 7: 08072102 Edwin.rtf - 7:3 [Agile System Development Lifec..] (35:35) (Super)

Codes: [Agile System Development Lifec..]

Code: Alpha testing {1-0}

P 9: 08081800 Kabelo.rtf - 9:19 [So with alpha, you'll use your..] (41:41) (Super)

Codes: [Alpha testing]

Code: Ambiguity on methodology that captures semantics {1-1}~

P 3: 07111501 Magda.rtf - 3:10 [I think if you really, you do ..] (57:57) (Super)

Codes: [Ambiguity on methodology that captures semantics]

I think if you really, you do your ERs, your diagrams, and your dataflow diagrams, if you do that correctly and you explain it correctly to your user, I would say there could be semantics. Let me just think about it - that question. The object orientation, if you look at the use case diagrams, and you get the documentation of all the use cases, okay that could be more meaningful in a sense. There could be some semantics in there. Okay, I would say both but I would say the object orientation really could be more meaningful.

Code: band width value analysis, {1-0}

P 9: 08081800 Kabelo.rtf - 9:8 [band width value analysis,] (9:9) (Super)

Codes: [band width value analysis,]

Code: Beta testing {1-0}

P 9: 08081800 Kabelo.rtf - 9:20 [beta is involved - whoever is ..] (41:41) (Super)
Codes: [Beta testing]

Code: brainstorming forum {1-1}

P 1: 07110506 Ernest.rtf - 1:10 [brainstorming forum] (27:27) (Super)
Codes: [brainstorming forum]

brainstorming forum

Code: bundle value analysis {1-0}

P 9: 08081800 Kabelo.rtf - 9:55 [bundle value analysis] (173:173) (Super)
Codes: [bundle value analysis]

Code: Business analyst as a project manager {1-0}

P 9: 08081800 Kabelo.rtf - 9:43 [in terms of Ned Bank, the busi..] (103:103) (Super)
Codes: [Business analyst as a project manager]

Code: business analysts {3-1}~

P 3: 07111501 Magda.rtf - 3:18 [business analysts] (84:84) (Super)
Codes: [business analysts]

business analysts

P 7: 08072102 Edwin.rtf - 7:9 [The business analyst needs to ..] (65:65) (Super)
Codes: [business analysts]

P 9: 08081800 Kabelo.rtf - 9:4 [business analyst] (9:9) (Super)
Codes: [business analysts]

Code: business requirement specifica.. {1-0}~

P 9: 08081800 Kabelo.rtf - 9:32 [Meaning before we can, even st..] (79:79) (Super)

Codes: [business requirement specifica..]

Code: Business Requirements Document {1-0}

P 9: 08081800 Kabelo.rtf - 9:3 [Whereby we'll take the busines..] (7:7) (Super)
Codes: [Business Requirements Document]

Code: business sponsors {1-1}

P 2: 07111500 Goede.rtf - 2:29 [business sponsors] (103:103) (Super)
Codes: [business sponsors]

business sponsors

Code: Capturing Human Behavior {2-2}

P 1: 07110506 Ernest.rtf - 1:53 [But it should take a much long..] (143:143) (Super)

Codes: [Capturing Human Behavior] [Humanist Requirement for Software Development]

But it should take a much longer study where you're trying to marry all those different fields because if you are going to do such a task, you need to know how people behave and that's a different field from IT. So the behaviour of human beings in different environments - how do they do that.

P 1: 07110506 Ernest.rtf - 1:54 [Now that they can solve a good..] (147:147) (Super)

Codes: [Capturing Human Behavior]

Now that they can solve a good number of those problems, we now expect them to help us in thinking as well, so that we have a thinking machine that can solve some of your thinking problems. In that area, that's where I would say we are lacking. We don't have the skills at the moment to actually come up with systems like that. The reason being that the human brain itself and the behaviour of the human being, those two issues are still grey even to those who are specialists in those areas.

Code: Capturing Pragmatic Knowledge {3-1}~

P 1: 07110506 Ernest.rtf - 1:1 [there are different approaches..] (23:23) (Super)

Codes: [Capturing Pragmatic Knowledge]

there are different approaches that people have tried to come up with that will assist in terms of capturing that kind of pragmatic knowledge, if you want to call it. Actually some people call it tacit knowledge. So one approach that should be used to actually capture that kind of knowledge, would be to allow in the systems development process a forum or some place within the process where that knowledge can be exchanged within the individuals who are involved in the development team.

P 1: 07110506 Ernest.rtf - 1:7 [In Extreme programming you hav..] (25:25) (Super)

Codes: [Capturing Pragmatic Knowledge] [Knowledge Sharing]

In Extreme programming you have a practice that is called Pair Programming. In Pair Programming you put two people - two programmers - on one machine and that way, that knowledge which can not be written down but which is in those developers' minds, can actually be exchanged as they do the development.

P 1: 07110506 Ernest.rtf - 1:9 [So in these informal meetings ..] (27:27) (Super)

Codes: [Capturing Pragmatic Knowledge] [Knowledge Sharing]

So in these informal meetings people discuss the technology, discuss the process and the type of work that they're doing. And you task especially the upcoming youngsters within the organisation to actually take down the information that people are bringing. And you also ask them to access some journals and so forth, and try to relate that information to the standard practices that people are following.

Code: Capturing Semantics Issue {3-0}

P 1: 07110506 Ernest.rtf - 1:57 [So that is exactly the demise ..] (155:155) (Super)

Codes: [Capturing Semantics Issue] [Language Limitations]

Memos: [ME - 11/13/08 [5]]

So that is exactly the demise that we have as software developers, that if we try to go into the details of the softer side - the non-formal parts of the system - we find that we have limitations in terms of the language. Because

whatever language you are using, when I try to use now the computer language to express that, it might not represent what you are talking about. So it is really the issues of language that should be dealt with and so forth.

P 3: 07111501 Magda.rtf - 3:37 [Object-orientation, I think, i..] (232:232) (Super)
Codes: [Capturing Semantics Issue] [object-oriented approach,]

Object-orientation, I think, if you look at the instantiation of a class, then I would say that could - object would be the - ja, the semantics behind the design. You design the class and then an instantiation of the class, the object would be the semantics.

P 9: 08081800 Kabelo.rtf - 9:31 [you get that from conceptual, ..] (69:69) (Super)
Codes: [Capturing Semantics Issue]

Code: Case Tool {1-0}

P 5: 07121401-Barry Myburgh Corrected.rtf - 5:19 [If the customer that you are b..] (71:71) (Super)
Codes: [Case Tool] [software engineering environme..]

If the customer that you are busy working for is the kind of customer who understands the importance of giving you enough time to do the job properly, then the use of case tools can greatly improve the software development process.

Code: Challenge for Software development {1-0}

P 5: 07121401-Barry Myburgh Corrected.rtf - 5:9 [And therefore I suppose I do a..] (27:27) (Super)
Codes: [Challenge for Software development] [Lack of humanist understanding]

And therefore I suppose I do agree that they're not programming them correctly but I'm not sure that it is possible to get that point where the computer can be as capable of understanding human speech the way that human beings can understand it. I don't necessarily think that it's an achievable goal.

Code: change control {1-0}

P 5: 07121401-Barry Myburgh Corrected.rtf - 5:37 [In a controlled quality enviro..] (125:125) (Super)
Codes: [change control]

In a controlled quality environment I apply change control. I go through all the mechanisms of managing change requests, evaluating the impact of the change, getting the customer to either agree to postpone the change, or pay more money for it, or to take more time for it or whatever. So change control is the answer.

Code: Collaborative Approach {1-0}

P 5: 07121401-Barry Myburgh Corrected.rtf - 5:17 [Obviously the most effective w..] (57:57) (Super)
Codes: [Collaborative Approach] [User Involvement]

Obviously the most effective way of trying to ensure that is if we have the client participate in the activities of the project. In other words, a collaborative approach where the client is really a part of the team.

Code: Communication Method {1-4}

P 8: 08072200 and 08072201-Sindiso.rtf - 8:18 [There's a system called Test D..] (74:74) (Super)

Codes: [Communication Method] [Test Director]

Code: Communication Problem {5-2}~

P 1: 07110506 Ernest.rtf - 1:31 [INT: I've met - in fact I've m..] (73:73) (Super)

Codes: [Communication Problem]

INT: I've met - in fact I've met two types of clients. One group of clients were clients who really don't know much about systems development. For example, one system we're actually developing, a school management system, and you're talking about people like school administrators and teachers and so forth; so most teachers would not know anything about systems developments. They will just tell you about their curriculum and the kind of day-to-day administrative tasks and chores that they actually have; and maybe try and interpret that into a system.

P 3: 07111501 Magda.rtf - 3:12 [And I - really I would argue t..] (66:66) (Super)

Codes: [Communication Problem]

And I - really I would argue that it's because the (client can't?) define the requirements correctly because we don't understand what exactly what this user wants.

P 9: 08081800 Kabelo.rtf - 9:44 [But now you find out that he m..] (105:105) (Super)

Codes: [Communication Problem] [Problems of not involving the users]

P 9: 08081800 Kabelo.rtf - 9:49 [But now in terms of communicat..] (119:119) (Super)

Codes: [Communication Problem] [Development Problem]

P 9: 08081800 Kabelo.rtf - 9:50 [So meaning communication, the ..] (123:123) (Super)

Codes: [Communication Problem]

Code: Communication problem between the conceptual and physical design. {1-1}~

P 3: 07111501 Magda.rtf - 3:36 [I would say maybe some of the ..] (220:220) (Super)

Codes: [Communication problem between the conceptual and physical design.]

I would say maybe some of the environments can handle that but (chuckling) sometimes it's difficult. It's up to a certain stage. Once you've generated your physical database, it's difficult. So I agree, it could help but I would first of all really test the system or - no, not the system - the environment to see that it's done, it can handle it.

Code: Communication Technique {4-18}

P 8: 08072200 and 08072201-Sindiso.rtf - 8:21 [The first one of course is jus..] (80:80) (Super)

Codes: [Communication Technique]

08072201-Sindiso.rtf

P 8: 08072200 and 08072201-Sindiso.rtf - 8:23 [We also have meeting sessions ..] (80:80) (Super)

Codes: [Communication Technique]

P 9: 08081800 Kabelo.rtf - 9:35 [Okay we use your normal UML to..] (87:87) (Super)

Codes: [Communication Technique]

P 9: 08081800 Kabelo.rtf - 9:38 [In Nedbank they were using Mic..] (93:93) (Super)

Codes: [Communication Technique] [Microsoft SharePoint]

Code: Communication Tool {1-2}

P 8: 08072200 and 08072201-Sindiso.rtf - 8:20 [emails] (80:80) (Super)

Codes: [Communication Tool]

Code: component development {1-1}~

P 3: 07111501 Magda.rtf - 3:29 [component development] (172:172) (Super)

Codes: [component development]

component development

Code: Concept negotiation Technique {3-1}~

P 1: 07110506 Ernest.rtf - 1:14 [So instead of people being stu..] (33:33) (Super)

Codes: [Concept negotiation Technique] [Language Community]

Memos: [ME - 11/12/08 [1]]

So instead of people being stuck to those technical terms and so forth, they come up with some naming convention that is common to everyone. And that naming convention brings everyone to the same language, despite the differences and so forth.

P 5: 07121401-Barry Myburgh Corrected.rtf - 5:27 [the reality centre and modelli..] (101:101) (Super)

Codes: [Concept negotiation Technique] [Language Requirement]

the reality centre and modelling things with everybody busily participating in the modelling that takes place, that to me is a very effective way of doing that mediation. Because if you are busy developing something like a mind map, you know, you type in the words that you think describe the point that has been made and the people who had made the point immediately see the words you use.

P 8: 08072200 and 08072201-Sindiso.rtf - 8:48 [Okay, how we do it at the mome..] (144:144) (Super)

Codes: [Concept negotiation Technique]

Code: configurable systems {1-0}~

P 5: 07121401-Barry Myburgh Corrected.rtf - 5:23 [configurable systems] (87:87) (Super)

Codes: [configurable systems]

configurable systems

Code: contingency approach, {1-1}~

P 3: 07111501 Magda.rtf - 3:7 [I would advocate a contingency..] (37:37) (Super)

Codes: [contingency approach,]

I would advocate a contingency approach, meaning I would first of all go and look at the type of project with the type of system that I would have to develop. Look at the characteristics of this system and then I would decide. Say for example it's a life critical system; you have to develop something for people in the hospitals or so. Then I would go for the structured approach, mainly because it's very, very focused; you know exactly what to do next. I would adjust the structured approach a bit to make it iterative, so you can go back to previous stages if you think there was a mistake there and because of the documentation. So then I would advocate that.

Code: Controlled quality approach requirements {1-0}

P 5: 07121401-Barry Myburgh Corrected.rtf - 5:33 [So if I'm in a situation where..] (115:115) (Super)
Codes: [Controlled quality approach requirements]

So if I'm in a situation where I have to - facing penalties of something - if I have to deliver within budget on time, I will demand to follow a controlled quality approach. And then that means I must be given sufficient time to have done sufficient analysis prior to the contract so that I really understand what it is I have to do, and can I plan accordingly, and I can form a team of people who can deliver on those requirements.

Code: controlled quality" {1-0}

P 5: 07121401-Barry Myburgh Corrected.rtf - 5:31 [The one I call "controlled qua..] (113:113) (Super)
Codes: [controlled quality"] [waterfall approach]

The one I call "controlled quality"; the other one I call "crafted quality". Now crafted quality is very much what most people call agile (inaudible - cross-talking) okay; and controlled quality is very much the kind-of conventional waterfall lifecycle approach.

Code: crafted quality" {1-0}

P 5: 07121401-Barry Myburgh Corrected.rtf - 5:32 [crafted quality". Now crafted ..] (113:113) (Super)
Codes: [Agile Approach Concept] [crafted quality"]

crafted quality". Now crafted quality is very much what most people call agile (inaudible - cross-talking) okay; and controlled quality is very much the kind-of conventional waterfall lifecycle approach.

Code: data-orientated paradigm {1-2}

P 3: 07111501 Magda.rtf - 3:2 [And the data-orientated paradi..] (25:25) (Super)
Codes: [data-orientated paradigm]

And the data-orientated paradigm is where they say [that] data stays constant, so we should rather focus on the data instead of the process, it would make our lives much easier.

Code: decision-logic-table {1-0}

P 9: 08081800 Kabelo.rtf - 9:11 [decision-logic-table] (9:9) (Super)
Codes: [decision-logic-table]

Code: Developer Understanding of Business Model {2-1}~

P 1: 07110506 Ernest.rtf - 1:38 [And besides that, when you do ..] (103:103) (Super)
Codes: [Developer Understanding of Business Model] [Knowledge Sharing]
Memos: [Studying the business environment]

And besides that, when you do such workshops, you are not just aiming at them understanding the system but you're also aiming at the developer's understanding of the business model that the organisation is using.

P 3: 07111501 Magda.rtf - 3:20 [So, once again - so you walk i..] (108:108) (Super)

Codes: [Developer Understanding of Business Model]

So, once again - so you walk into a company and you have to find out what these assumptions are. That's the way. Once again, a deeper of knowledge of business and communication. I think that's a big problem for consultants, if they go into the business and they just have to do consultancy (inaudible - noise) straight answer in my opinion would be communication and hang on a bit, just get to know the business. And after you've done that for a period, you can start with your work. But I don't think there's a luxury for that.

Code: Development Approach {1-8}

P 8: 08072200 and 08072201-Sindiso.rtf - 8:13 [I would prefer, ... a method in ..] (68:68) (Super)

Codes: [Development Approach] [New Development Approach]

Code: Development Method {0-13}

Code: Development Problem {9-5}~

P 1: 07110506 Ernest.rtf - 1:21 [And then you find that we don't ..] (55:55) (Super)

Codes: [Development Problem]

And then you find that we don't develop systems that actually directly mix the requirements of that particular individual because there's a gap between the understanding of how the system works from the core of the system - the methods and the procedures and so forth - and the actual interface that the user is actually using.

P 1: 07110506 Ernest.rtf - 1:22 [So you have the business that ..] (57:57) (Super)

Codes: [Development Problem] [Translation Medium Requirement]

So you have the business that side, you have got the interface which the user is using, and you have the technical details of the system itself. So those are three different levels and you find that at those three different levels the user understands the business side and the interface, but does not really understand the technical details but the developer who is the one who has the technical details. So you find that that way now, those requirements will always be presented in the way that the technical guy understands which puts the user at a disadvantage.

P 3: 07111501 Magda.rtf - 3:13 [And I - really I would argue t..] (66:66) (Super)

Codes: [Development Problem]

Memos: [Requirements problem]

And I - really I would argue that it's because the (client can't?) define the requirements correctly because we don't understand what exactly what this user wants. So if we put more effort into requirements gathering, I think you will see that problem - I wouldn't say it'll be solved - but much better. I think we are so much in haste to get the product delivered, that we don't even bother to look at the requirements.

P 3: 07111501 Magda.rtf - 3:16 [I don't know about you Nehemia..] (72:72) (Super)

Codes: [Development Problem]

Memos: [Lack of trust]

I don't know about you Nehemiah but - there's sort-of a hostility between users and system developers. The users, they don't trust the developers and the developers talk down on the user. Okay, and that's something that's still alive and well and that bothers me. We are partners ?

P 8: 08072200 and 08072201-Sindiso.rtf - 8:26 [Because right now we can't do ..] (88:88) (Super)

Codes: [Development Problem]

P 8: 08072200 and 08072201-Sindiso.rtf - 8:39 [there are many times where the..] (122:122) (Super)
Codes: [Development Problem]

P 8: 08072200 and 08072201-Sindiso.rtf - 8:41 [Many a time analysts do not re..] (124:124) (Super)
Codes: [Development Problem]

P 9: 08081800 Kabelo.rtf - 9:49 [But now in terms of communicat..] (119:119) (Super)
Codes: [Communication Problem] [Development Problem]

P 9: 08081800 Kabelo.rtf - 9:51 [Because with the current - set..] (125:125) (Super)
Codes: [Development Problem]

Code: Development Technique {0-10}

Code: Development Tool {0-3}

Code: differential testing, {1-0}

P 8: 08072200 and 08072201-Sindiso.rtf - 8:55 [differential testing,] (150:150) (Super)
Codes: [differential testing,]

Code: Discussion Forum {2-1}~

P 1: 07110506 Ernest.rtf - 1:3 [So one approach that should be..] (23:23) (Super)
Codes: [Discussion Forum]

So one approach that should be used to actually capture that kind of knowledge, would be to allow in the systems development process a forum or some place within the process where that knowledge can be exchanged within the individuals who are involved in the development team.

P 1: 07110506 Ernest.rtf - 1:12 [So these whiteboards would be ..] (31:31) (Super)
Codes: [Discussion Forum] [Knowledge Sharing]

So these whiteboards would be - that's where people would just write those terms that they are used to, that they normally use. And then the flyers would actually - when people have a meeting they talk and so forth - then you stick those along the corridors in that organisation. So that as you walk up and down, you actually see those common terms and so forth

Code: document of inputs {1-0}~

P 8: 08072200 and 08072201-Sindiso.rtf - 8:47 [document of inputs] (144:144) (Super)
Codes: [document of inputs]

Code: Documentation as a communication tool/method {1-0}~

P 5: 07121401-Barry Myburgh Corrected.rtf - 5:16 [But of course that is appropri..] (53:53) (Super)
Codes: [Documentation as a communication tool/method]

But of course that is appropriate during the formatory (sp) stages of the project when you are busy driving out the understanding and so on. If you're getting more to the formal stages where you are saying [that] here is an artefact or a deliverable that is being packaged, and now needs to be reviewed and accepted, that's a totally different matter.

There I stick to the classical methods of distributing the documentation; everybody walks through the documentation and reads and approves what they've seen.

Code: Dooijeweert aspects {1-1}

P 2: 07111500 Goede.rtf - 2:33 [Dooijeweert aspects] (111:111) (Super)

Codes: [Dooijeweert aspects]

Dooijeweert aspects

Code: Emphasis of Formal part {2-0}

P 5: 07121401-Barry Myburgh Corrected.rtf - 5:11 [you know, in our software deve..] (39:39) (Super)

Codes: [Emphasis of Formal part]

you know, in our software development processes we tend to focus almost exclusively on the formal part. I've certainly not been involved in any information system development project where we've tried to create a computer that can sort-of adapt it's response to different needs without having to be changed. So - I mean, is that what you're sort-of suggesting? It's almost an artificial intelligence kind-of -

P 5: 07121401-Barry Myburgh Corrected.rtf - 5:12 [The systems that I'm involved ..] (43:43) (Super)

Codes: [Emphasis of Formal part]

The systems that I'm involved with - have been involved with, have always been ones where we've said - we've specified the formal parts on how this thing must work and we build it to work that way, and that's what the customer wants.

Code: equivalence {2-0}

P 9: 08081800 Kabelo.rtf - 9:9 [equivalence] (9:9) (Super)

Codes: [equivalence]

P 9: 08081800 Kabelo.rtf - 9:56 [use the classic equivalency.] (173:173) (Super)

Codes: [equivalence]

Code: ERDs {1-2}

P 2: 07111500 Goede.rtf - 2:14 [That's why ERDs are so difficu..] (55:55) (Super)

Codes: [ERDs]

That's why ERDs are so difficult and not a very good tool to use, except in the physical design. It's not a requirements validation tool for end-users.

Code: Evolvable Software Products Concept {4-1}

P 1: 07110506 Ernest.rtf - 1:50 [So basically evolving systems ..] (135:135) (Super)

Codes: [Evolvable Software Products Concept]

So basically evolving systems are systems that don't remain doing what they were originally meant to do but these are systems where, based on the outputs that you get and the comments that the clients give you, you actually regulate - you maintain the system - you implement some changes in the system and allow that system to be adapted to the new environment that you have.

P 2: 07111500 Goede.rtf - 2:26 [this incremental development i..] (87:87) (Super)
Codes: [Evolvable Software Products Concept]

this incremental development idea that we create a software package or product that is delivered in increments; and that you don't gather all the requirements (indiscernible - audio quality) from the starting step (indiscernible - audio quality) as you deliver, get more requirements and feedback, so that you have a better chance of designing a product that suits the real needs of the (indiscernible - audio quality).

P 3: 07111501 Magda.rtf - 3:30 [It grows over time. Evolve, ma..] (188:188) (Super)
Codes: [Evolvable Software Products Concept]

It grows over time. Evolve, maybe I can draw a picture. I don't know if that's (inaudible - noise). What I would say is, you have your problem. Say for example that's your problem. And then I would normally - evolutionary development, I would divide it into certain chunks and then I would start. I would only implement that one - develop and implement. If it's spelled correctly, implement, I don't know. And then afterward you add something on. So that one would be in the organisation, they would use that. And then you build the second part, and you develop (indiscernible - audio quality) and implement. This is how I see evolutionary development, that with time it grows. So it's like incremental development, I would say. (indiscernible - audio quality) that's how I understand it.

P 5: 07121401-Barry Myburgh Corrected.rtf - 5:22 [Well that's very often the sit..] (83:83) (Super)
Codes: [Evolvable Software Products Concept]

Well that's very often the situation where we don't have time to do the job thoroughly the first time we do it, so we try to deliver the first working version of the system as quickly as possible, and then we use that as a basis for evolving the system into the eventual scope and functionality that the customer really needs.

Code: Evolving Product development Technique {3-2}~

P 1: 07110506 Ernest.rtf - 1:27 [So you're breaking down the re..] (65:65) (Super)
Codes: [Evolving Product development Technique]

So you're breaking down the requirements into those small user stories. Then you take those user stories now and ask the guy who is going to develop to now try and estimate how long it's going to take to actually develop that. Then you are coming up with some design already as you do that.

P 3: 07111501 Magda.rtf - 3:31 [incremental development] (188:188) (Super)
Codes: [Evolving Product development Technique]

incremental development

P 3: 07111501 Magda.rtf - 3:32 [XP, the methodology called XP?..] (192:192) (Super)
Codes: [Evolving Product development Technique] [Extreme programming]

XP, the methodology called XP? Xtreme Programming.

Code: Extreme programming {3-4}~

P 1: 07110506 Ernest.rtf - 1:5 [Extreme programming] (25:25) (Super)
Codes: [Extreme programming]

Extreme programming

P 1: 07110506 Ernest.rtf - 1:15 [Extreme Programming -] (33:33) (Super)

Codes: [Extreme programming]

Extreme Programming -

P 3: 07111501 Magda.rtf - 3:32 [XP, the methodology called XP?..] (192:192) (Super)

Codes: [Evolving Product development Technique] [Extreme programming]

XP, the methodology called XP? Xtreme Programming.

Code: First step in developing software {1-0}

P 9: 08081800 Kabelo.rtf - 9:47 [I think if we can get the righ..] (117:117) (Super)

Codes: [First step in developing software]

Code: functional analyst, {1-0}

P 9: 08081800 Kabelo.rtf - 9:5 [functional analyst,] (9:9) (Super)

Codes: [functional analyst,]

Code: Functional Design Document {1-0}

P 9: 08081800 Kabelo.rtf - 9:33 [FDD] (51:51) (Super)

Codes: [Functional Design Document]

Code: functional testing {2-0}

P 8: 08072200 and 08072201-Sindiso.rtf - 8:53 [functional testing] (148:148) (Super)

Codes: [functional testing]

P 9: 08081800 Kabelo.rtf - 9:14 [We'll test the functionality b..] (11:11) (Super)

Codes: [functional testing]

Code: heuristic systems {1-0}~

P 5: 07121401-Barry Myburgh Corrected.rtf - 5:21 [heuristic systems that could l..] (79:79) (Super)

Codes: [Adaptive Systems] [heuristic systems]

heuristic systems that could learn from the way that a person uses the system, and sort-of adapt its response to become more and more inline with how that person uses it.

Code: Humanist Requirement for Software Development {3-0}

P 1: 07110506 Ernest.rtf - 1:51 [Because software development i..] (139:139) (Super)

Codes: [Humanist Requirement for Software Development]

Because software development is not like other IT fields, like traditional engineering and so forth. You find that you can't come up with blueprints and put them there and say [that] people are going to develop according to this, and they follow that because basically things are based on the human brain, it's more like an art.

P 1: 07110506 Ernest.rtf - 1:53 [But it should take a much long..] (143:143) (Super)

Codes: [Capturing Human Behavior] [Humanist Requirement for Software Development]

But it should take a much longer study where you're trying to marry all those different fields because if you are going to do such a task, you need to know how people behave and that's a different field from IT. So the behaviour of human beings in different environments - how do they do that.

P 3: 07111501 Magda.rtf - 3:5 [we shouldn't forget is paradig..] (27:27) (Super)
Codes: [Humanist Requirement for Software Development]

we shouldn't forget is paradigms where they focus on people, people-orientated paradigms. We say, in the end, all the work we do, the systems we develop, the products we develop, is there to benefit people. So you should really be in contact with your users and so on. So that seems more it's a social aspect - your systems development. And I think that's where your user interface; maybe it could link to that.

Code: impact analysis {1-0}

P 7: 08072102 Edwin.rtf - 7:1 [impact analysis sort-of on thi..] (25:25) (Super)
Codes: [impact analysis]

Code: incremental approach {1-0}

P 9: 08081800 Kabelo.rtf - 9:16 [So meaning users will have to ..] (23:23) (Super)
Codes: [incremental approach]

Code: Incremental development {2-2}

P 2: 07111500 Goede.rtf - 2:18 [Incremental development helps ..] (63:63) (Super)
Codes: [Incremental development]

Incremental development helps a lot to do delivery in smaller pieces, then if you go wrong you don't have to backtrack the whole system. So an iteration and an incremental development helps a lot.

P 9: 08081800 Kabelo.rtf - 9:17 [So meaning users will have to ..] (23:23) (Super)
Codes: [Incremental development]

Code: information engineering {1-1}~

P 3: 07111501 Magda.rtf - 3:26 [information engineering] (160:160) (Super)
Codes: [information engineering]

information engineering

Code: inspections {1-0}

P 9: 08081800 Kabelo.rtf - 9:28 [inspections] (53:53) (Super)
Codes: [inspections]

Code: integration testing {1-0}

P 8: 08072200 and 08072201-Sindiso.rtf - 8:51 [integration testing] (148:148) (Super)
Codes: [integration testing]

Code: interactive development {1-1}

P 1: 07110506 Ernest.rtf - 1:41 [interactive development”, wher..] (111:111) (Super)
Codes: [interactive development]

interactive development”, where you develop your system in small iterations, and at the end of each iteration, you actually deliver a part of the system that is actually working.

Code: intuitive system {1-0}

P 8: 08072200 and 08072201-Sindiso.rtf - 8:15 [So it’s more of an intuitive s..] (70:70) (Super)
Codes: [intuitive system]

Code: Issues to consider when choosing a methodology {1-0}

P 9: 08081800 Kabelo.rtf - 9:48 [I think if we can get the righ..] (117:117) (Super)
Codes: [Issues to consider when choosing a methodology]

Code: Jad sessions {1-0}

P 9: 08081800 Kabelo.rtf - 9:25 [Jad sessions] (51:51) (Super)
Codes: [Jad sessions]

Code: Knowledge Base {1-0}

P 5: 07121401-Barry Myburgh Corrected.rtf - 5:25 [I mean to give a - probably no..] (95:95) (Super)
Codes: [Knowledge Base]

I mean to give a - probably not a very academic answer but in my personal life as at school - you know, pretty much as an independent consultant and contractor, I just keep everything that I’ve done and re-use whatever wherever I can. So I mean, it’s actually not a matter of knowledge base that you can draw on and just re-use stuff and repackage it and stuff like that. So I mean that’s really the short answer but it all depends on my memory of course, and my search engine’s capability of finding the things at the right time. I mean, I literally on my notebook computer have got all the work I have done for the last 15 years is on that computer. So if I can remember that I’ve done it before for somebody else, and my desktop search engine can find it, then I’ll re-use it. If I can’t find it I don’t re-use it. So it’s really not a very good answer to your question.

Code: Knowledge reuse {1-0}~

P 2: 07111500 Goede.rtf - 2:27 [We call it experience. One thi..] (91:91) (Super)
Codes: [Knowledge reuse]

We call it experience. One thing one can say is that involvement of the right people from the start. If you’ve written a system and you - normally when it fails it is kind-of because that the right people was not involved from the company. That’s my experience, that the right people were not involved from the start and then that kind of experience I will take along and do a readiness assessment before I do anything else, and get the right people involved before I start working again on a new project.

Code: Knowledge Sharing {6-0}

P 1: 07110506 Ernest.rtf - 1:7 [In Extreme programming you hav..] (25:25) (Super)
Codes: [Capturing Pragmatic Knowledge] [Knowledge Sharing]

In Extreme programming you have a practice that is called Pair Programming. In Pair Programming you put two people - two programmers - on one machine and that way, that knowledge which can not be written down but which is in those developers' minds, can actually be exchanged as they do the development.

P 1: 07110506 Ernest.rtf - 1:9 [So in these informal meetings ..] (27:27) (Super)
Codes: [Capturing Pragmatic Knowledge] [Knowledge Sharing]

So in these informal meetings people discuss the technology, discuss the process and the type of work that they're doing. And you task especially the upcoming youngsters within the organisation to actually take down the information that people are bringing. And you also ask them to access some journals and so forth, and try to relate that information to the standard practices that people are following.

P 1: 07110506 Ernest.rtf - 1:12 [So these whiteboards would be ..] (31:31) (Super)
Codes: [Discussion Forum] [Knowledge Sharing]

So these whiteboards would be - that's where people would just write those terms that they are used to, that they normally use. And then the flyers would actually - when people have a meeting they talk and so forth - then you stick those along the corridors in that organisation. So that as you walk up and down, you actually see those common terms and so forth

P 1: 07110506 Ernest.rtf - 1:38 [And besides that, when you do ..] (103:103) (Super)
Codes: [Developer Understanding of Business Model] [Knowledge Sharing]
Memos: [Studying the business environment]

And besides that, when you do such workshops, you are not just aiming at them understanding the system but you're also aiming at the developer's understanding of the business model that the organisation is using.

P 1: 07110506 Ernest.rtf - 1:58 [But what usually happens is th..] (161:161) (Super)
Codes: [Knowledge Sharing] [Language Community]
Memos: [ME - 11/13/08 [6]]

But what usually happens is that when you get to an organisation, you need to understand their language, the way they talk about things, whether they are technical or non-technical but for the purposes of your requirements. The way the information is represented will sometimes differ from other organisations even in the same field.

P 8: 08072200 and 08072201-Sindiso.rtf - 8:45 [So it is crucial that as a tea..] (136:136) (Super)
Codes: [Knowledge Sharing]

Code: Lack of humanist understanding {1-0}

P 5: 07121401-Barry Myburgh Corrected.rtf - 5:9 [And therefore I suppose I do a..] (27:27) (Super)
Codes: [Challenge for Software development] [Lack of humanist understanding]

And therefore I suppose I do agree that they're not programming them correctly but I'm not sure that it is possible to get that point where the computer can be as capable of understanding human speech the way that human beings can understand it. I don't necessarily think that it's an achievable goal.

Code: Language Community {7-0}~

P 1: 07110506 Ernest.rtf - 1:14 [So instead of people being stu..] (33:33) (Super)
Codes: [Concept negotiation Technique] [Language Community]
Memos: [ME - 11/12/08 [1]]

So instead of people being stuck to those technical terms and so forth, they come up with some naming convention that is common to everyone. And that naming convention brings everyone to the same language, despite the differences and so forth.

P 1: 07110506 Ernest.rtf - 1:28 [So sitting together there is a..] (67:67) (Super)
Codes: [Language Community] [User Involvement]

So sitting together there is a very important aspect or practice of eliciting the requirements or systems development. You are saying that these three guys should be together. You don't want the systems analyst to meet some guys there, get some documents signed and then go and shift that information into the programme, but you want these three levels to actually sit together. The user must be the centre of all those things.

P 1: 07110506 Ernest.rtf - 1:58 [But what usually happens is th..] (161:161) (Super)
Codes: [Knowledge Sharing] [Language Community]
Memos: [ME - 11/13/08 [6]]

But what usually happens is that when you get to an organisation, you need to understand their language, the way they talk about things, whether they are technical or non-technical but for the purposes of your requirements. The way the information is represented will sometimes differ from other organisations even in the same field.

P 3: 07111501 Magda.rtf - 3:14 [And there should be users at o..] (72:72) (Super)
Codes: [Language Community]

And there should be users at our side when we develop these requirements, and we should give them examples.

P 5: 07121401-Barry Myburgh Corrected.rtf - 5:15 [in fact one of my customers a ..] (53:53) (Super)
Codes: [Language Community] [Need for a communication medium.] [Reality Centre]

in fact one of my customers a few years back, they had a centre - a room which was dedicated to sharing information, they called it a reality centre. And that sort-of supports my thinking as well, where the best way to communicate is to use a data projector, you use some kind of tool. I'm a great believer in using mind-mapping as a tool to share thoughts and to achieve consensus. So I would typically project on the screen as the mind-map develops, as I'm engaging with the people that have to be part of the buy-in in the model that is developed, and they participate in the development of the model. So it's very much like in a real JAD session, where you use some tools to develop a model and you involve everyone in visually in this kind of reality centre concept of participating in the development of the model.

P 5: 07121401-Barry Myburgh Corrected.rtf - 5:26 [I tried to be sensitive to the..] (99:99) (Super)
Codes: [Language Community]

I tried to be sensitive to the fact that the words people use don't necessarily mean the same thing to different people. So I'm always on the lookout for that possibility and then just use effective listening techniques - you know, reflection, confirmation of understanding and so on, to try to ensure that we have actually got the same meaning although we started out using different words.

P 8: 08072200 and 08072201-Sindiso.rtf - 8:44 [So it is quite important that,..] (134:134) (Super)
Codes: [Language Community]

Code: language database {1-0}~

P 1: 07110506 Ernest.rtf - 1:59 [language database] (163:163) (Super)
Codes: [language database]

language database

Code: Language Limitations {2-0}~

P 1: 07110506 Ernest.rtf - 1:56 [For example, when somebody exp..] (155:155) (Super)

Codes: [Language Limitations]

Memos: [ME - 11/13/08 [4]]

For example, when somebody expresses their idea, the limitations of that expression or that representation, are within the language that they are using.

P 1: 07110506 Ernest.rtf - 1:57 [So that is exactly the demise ..] (155:155) (Super)

Codes: [Capturing Semantics Issue] [Language Limitations]

Memos: [ME - 11/13/08 [5]]

So that is exactly the demise that we have as software developers, that if we try to go into the details of the softer side - the non-formal parts of the system - we find that we have limitations in terms of the language. Because whatever language you are using, when I try to use now the computer language to express that, it might not represent what you are talking about. So it is really the issues of language that should be dealt with and so forth.

Code: Language Requirement {4-0}

P 1: 07110506 Ernest.rtf - 1:55 [because the language is the on..] (153:153) (Super)

Codes: [Language Requirement] [New Development Approach]

Memos: [ME - 11/13/08 [3]]

because the language is the only solid thing or it's the only output that you get from a human being that lets you know what is happening in their brain. And if you want to deal with the informal part of the system, you are really looking at the softer issues, the human issues of software development. The behaviour of human beings and their languages would be the centre that people have to deal with in order to understand.

P 3: 07111501 Magda.rtf - 3:17 [Plain common language; not tec..] (84:84) (Super)

Codes: [Language Requirement]

Plain common language; not technical jargon. Sometimes we don't even - we are not familiar with our own language, the terms we use, the slang that we use. So we should really, really use just plain common language at the level that the user can understand.

P 5: 07121401-Barry Myburgh Corrected.rtf - 5:27 [the reality centre and modelli..] (101:101) (Super)

Codes: [Concept negotiation Technique] [Language Requirement]

the reality centre and modelling things with everybody busily participating in the modelling that takes place, that to me is a very effective way of doing that mediation. Because if you are busy developing something like a mind map, you know, you type in the words that you think describe the point that has been made and the people who had made the point immediately see the words you use.

P 8: 08072200 and 08072201-Sindiso.rtf - 8:5 [But like I said, something bet..] (34:34) (Super)

Codes: [Language Requirement]

Code: last partitioning {1-0}

P 9: 08081800 Kabelo.rtf - 9:10 [last partitioning] (9:9) (Super)

Codes: [last partitioning]

Code: Link amongst stakeholders in software development {2-0}

P 9: 08081800 Kabelo.rtf - 9:45 [It goes with the methodology, ..] (109:109) (Super)
Codes: [Link amongst stakeholders in software development]

P 9: 08081800 Kabelo.rtf - 9:46 [So, it goes with the methodolo..] (113:113) (Super)
Codes: [Link amongst stakeholders in software development] [Need for
concept negotiation method]

Code: Link specifications to programming language {1-0}

P 8: 08072200 and 08072201-Sindiso.rtf - 8:38 [I personally feel, it's a good..] (122:122) (Super)
Codes: [Link specifications to programming language]

Code: load testing {1-0}

P 8: 08072200 and 08072201-Sindiso.rtf - 8:54 [load testing] (148:148) (Super)
Codes: [load testing]

Code: Managing Scope Creep {3-0}~

P 5: 07121401-Barry Myburgh Corrected.rtf - 5:38 [In a controlled quality enviro..] (125:125) (Super)
Codes: [Managing Scope Creep]

In a controlled quality environment I apply change control. I go through all the mechanisms of managing change requests, evaluating the impact of the change, getting the customer to either agree to postpone the change, or pay more money for it, or to take more time for it or whatever. So change control is the answer.

P 5: 07121401-Barry Myburgh Corrected.rtf - 5:39 [In an evolutionary adaptive - ..] (127:127) (Super)
Codes: [Managing Scope Creep]

In an evolutionary adaptive - no what - a crafted quality, agile world I probably manage it quite differently. I say to myself - and that's where the customer collaboration is so important because the customer must be a part of that decision to say "okay this change that we are now considering, is it more important than the objectives that we set for ourselves at the beginning of this timebox. And if we agree it is more important, then we add that to the scope of work and we'll probably accept at the same time that some of the other work that was intended will fall off the slate because there's just not time and resources to do it".

P 7: 08072102 Edwin.rtf - 7:11 [And one of the best things tha..] (65:65) (Super)
Codes: [Managing Scope Creep]

Code: Mechanistic View {0-2}

Code: meeting sessions {1-0}

P 8: 08072200 and 08072201-Sindiso.rtf - 8:22 [meeting sessions] (80:80) (Super)
Codes: [meeting sessions]

Code: Microsoft SharePoint {1-0}

P 9: 08081800 Kabelo.rtf - 9:38 [In Nedbank they were using Mic..] (93:93) (Super)

Codes: [Communication Technique] [Microsoft SharePoint]

Code: Mind Mapping {1-0}

P 9: 08081800 Kabelo.rtf - 9:36 [Else we'll use your mind maps ..] (87:87) (Super)

Codes: [Mind Mapping]

Code: Need a Variety of Development Tools {2-0}

P 1: 07110506 Ernest.rtf - 1:36 [that you are dealing with peop..] (97:97) (Super)

Codes: [Need a Variety of Development Tools]

Memos: [ME - 11/13/08]

that you are dealing with people and people are important in those systems, and what they know is also important. It means that if you try to come up with tools like those, you are threatening the development approach itself. The methodology is threatened because you are restricting to it a specific tool within the project. Remember the projects are different and the skill sets of people that you work with are different.

P 8: 08072200 and 08072201-Sindiso.rtf - 8:12 [I agree, it's basically the po..] (66:66) (Super)

Codes: [Need a Variety of Development Tools] [Need for a communication medium.] [Problems in Software development]

Code: Need for a communication medium. {3-0}~

P 1: 07110506 Ernest.rtf - 1:30 [One group of clients were clie..] (73:73) (Super)

Codes: [Need for a communication medium.]

Memos: [ME - 11/12/08 [8]]

One group of clients were clients who really don't know much about systems development.

P 5: 07121401-Barry Myburgh Corrected.rtf - 5:15 [in fact one of my customers a ..] (53:53) (Super)

Codes: [Language Community] [Need for a communication medium.] [Reality Centre]

in fact one of my customers a few years back, they had a centre - a room which was dedicated to sharing information, they called it a reality centre. And that sort-of supports my thinking as well, where the best way to communicate is to use a data projector, you use some kind of tool. I'm a great believer in using mind-mapping as a tool to share thoughts and to achieve consensus. So I would typically project on the screen as the mind-map develops, as I'm engaging with the people that have to be part of the buy-in in the model that is developed, and they participate in the development of the model. So it's very much like in a real JAD session, where you use some tools to develop a model and you involve everyone in visually in this kind of reality centre concept of participating in the development of the model.

P 8: 08072200 and 08072201-Sindiso.rtf - 8:12 [I agree, it's basically the po..] (66:66) (Super)

Codes: [Need a Variety of Development Tools] [Need for a communication medium.] [Problems in Software development]

Code: Need for a development tool {2-0}

P 2: 07111500 Goede.rtf - 2:24 [Firstly because I think it's m..] (75:75) (Super)

Codes: [Need for a development tool] [Requirements Repository]

Memos: [Requirements Repository]

Firstly because I think it's mostly - let me say, I don't think the end-user will benefit as much as the developer from doing it because you have one version of the truth of the requirement, you have one place - central repository -

where you can find the requirements, you can set up checklists to see if you have fulfilled the requirements, the requirements are (nearby??) but then the system must force the developers or the analysts to fill it in.

P 8: 08072200 and 08072201-Sindiso.rtf - 8:7 [So I mean the process of migra..] (40:40) (Super)
Codes: [Need for a development tool]

Code: Need for a language between analysis and design models {1-0}

P 8: 08072200 and 08072201-Sindiso.rtf - 8:9 [The conceptual view, usually i..] (57:57) (Super)
Codes: [Need for a language between analysis and design models]

Code: Need for concept negotiation method {3-0}

P 8: 08072200 and 08072201-Sindiso.rtf - 8:43 [“by interface, are you relatin..] (132:132) (Super)
Codes: [Need for concept negotiation method]

P 8: 08072200 and 08072201-Sindiso.rtf - 8:46 [At the bank we have this appli..] (138:138) (Super)
Codes: [Need for concept negotiation method]

P 9: 08081800 Kabelo.rtf - 9:46 [So, it goes with the methodolo..] (113:113) (Super)
Codes: [Link amongst stakeholders in software development] [Need for concept negotiation method]

Code: Need for documentation {1-0}~

P 8: 08072200 and 08072201-Sindiso.rtf - 8:34 [So in essence, our document, a..] (102:102) (Super)
Codes: [Need for documentation]

Code: Need to adapt systems to organizational requirements {1-0}

P 8: 08072200 and 08072201-Sindiso.rtf - 8:8 [You can have a different - say..] (46:47) (Super)
Codes: [Need to adapt systems to organizational requirements]

Code: Need to improve programming languages {5-0}~

P 8: 08072200 and 08072201-Sindiso.rtf - 8:1 [In essence, like you’re saying..] (23:23) (Super)
Codes: [Need to improve programming languages]

P 8: 08072200 and 08072201-Sindiso.rtf - 8:2 [So in a way it’s in the struct..] (25:25) (Super)
Codes: [Need to improve programming languages]

P 8: 08072200 and 08072201-Sindiso.rtf - 8:24 [So in a essence we need to get..] (86:86) (Super)
Codes: [Need to improve programming languages]

P 8: 08072200 and 08072201-Sindiso.rtf - 8:25 [first of all I believe we shou..] (88:88) (Super)
Codes: [Need to improve programming languages] [Requirements of a new development approach]

P 8: 08072200 and 08072201-Sindiso.rtf - 8:27 [So, first of all we need to de..] (90:90) (Super)
Codes: [Need to improve programming languages]

Code: Negating the analysis model characteristics {1-0}

P 8: 08072200 and 08072201-Sindiso.rtf - 8:11 [So, in essence, they tend to f..] (61:61) (Super)
Codes: [Negating the analysis model characteristics]

Code: New Development Approach {3-0}

P 1: 07110506 Ernest.rtf - 1:17 [So I think the best way to act..] (43:43) (Super)
Codes: [New Development Approach]
Memos: [ME - 11/12/08 [3]]

So I think the best way to actually represent that information will be to come up with some - not a framework - but maybe language of some sort that has the ability to actually express those concepts and to relate to technical issues that way. So I'm not very sure of what language you could actually use for that but I'm tending to think in terms of the approaches to development.

P 1: 07110506 Ernest.rtf - 1:55 [because the language is the on..] (153:153) (Super)
Codes: [Language Requirement] [New Development Approach]
Memos: [ME - 11/13/08 [3]]

because the language is the only solid thing or it's the only output that you get from a human being that lets you know what is happening in their brain. And if you want to deal with the informal part of the system, you are really looking at the softer issues, the human issues of software development. The behaviour of human beings and their languages would be the centre that people have to deal with in order to understand.

P 8: 08072200 and 08072201-Sindiso.rtf - 8:13 [I would prefer, ... a method in ..] (68:68) (Super)
Codes: [Development Approach] [New Development Approach]

Code: object-oriented approach, {6-5}~

P 1: 07110506 Ernest.rtf - 1:44 [object-oriented approach,] (117:117) (Super)
Codes: [object-oriented approach,]

object-oriented approach,

P 2: 07111500 Goede.rtf - 2:3 [Object-oriented programming to..] (19:19) (Super)
Codes: [object-oriented approach,]
Memos: [ME - 01/12/09 [1]]

Object-oriented programming to me works well in the technical applications. Perhaps it's not yet proven enough for me in today's software development, except perhaps for requirement analysis.

P 2: 07111500 Goede.rtf - 2:4 [object-oriented technique does..] (23:23) (Super)
Codes: [object-oriented approach,]

object-oriented technique does allow more for people to capture all their requirements, to capture non-technical things.

P 3: 07111501 Magda.rtf - 3:4 [object-orientated paradigm] (27:27) (Super)
Codes: [object-oriented approach,]

object-orientated paradigm

P 3: 07111501 Magda.rtf - 3:37 [Object-orientation, I think, i..] (232:232) (Super)
Codes: [Capturing Semantics Issue] [object-oriented approach,]

Object-orientation, I think, if you look at the instantiation of a class, then I would say that could - object would be the - ja, the semantics behind the design. You design the class and then an instantiation of the class, the object would be the semantics.

P 8: 08072200 and 08072201-Sindiso.rtf - 8:3 [The object oriented approach s..] (28:28) (Super)
Codes: [object-oriented approach,]

Code: object-oriented paradigm {1-0}

P 5: 07121401-Barry Myburgh Corrected.rtf - 5:4 [in principle the object-orient..] (17:17) (Super)
Codes: [object-oriented paradigm]

in principle the object-oriented paradigm lends itself to modelling reality; and from that perspective then I think, you know, in principle it should be better positioned to perhaps then capture those semantics as you described them.

Code: Object orientation versus structured {1-0}

P 3: 07111501 Magda.rtf - 3:11 [The object orientation, if you..] (57:57) (Super)
Codes: [Object orientation versus structured]

The object orientation, if you look at the use case diagrams, and you get the documentation of all the use cases, okay that could be more meaningful in a sense. There could be some semantics in there. Okay, I would say both but I would say the object orientation really could be more meaningful. I

Code: object oriented language {1-0}

P 8: 08072200 and 08072201-Sindiso.rtf - 8:30 [object oriented language] (96:96) (Super)
Codes: [object oriented language]

Code: Ontology Knowledge Base {1-0}

P 1: 07110506 Ernest.rtf - 1:61 [So what I'm saying is that you..] (165:165) (Super)
Codes: [Ontology Knowledge Base] [Process Ontology]
Memos: [ME - 11/13/08 [8]]

So what I'm saying is that you need some formal way where you can have a database that has some schemas of those words, of those concepts that we're talking about. For example, if we talk about a loan, all the financial institutions will have something called a loan. So you can have a loan schema, you find that your Nedbank, your ABSAs and Standard Banks might actually have different terms for your interest - for the interest rate, for whatever, for the borrowing rate or whatever - you know, those technical terms that they use. They might have different terms that actually mean the same thing, so your schema must be able to capture that - ensure that it is the same thing so that when you now check back and re-use it in different institutions you are not really changing much. It will come out there according to their language but you are still using the same thing

Code: Oracle Designer {1-1}~

P 3: 07111501 Magda.rtf - 3:24 [Oracle Designer] (148:148) (Super)
Codes: [Oracle Designer]

Oracle Designer

Code: Pair Programming {1-2}~

P 1: 07110506 Ernest.rtf - 1:6 [Pair Programming] (25:25) (Super)
Codes: [Pair Programming]

Pair Programming

Code: peer reviews. {2-0}

P 9: 08081800 Kabelo.rtf - 9:27 [peer reviews.] (53:53) (Super)
Codes: [peer reviews.]

P 9: 08081800 Kabelo.rtf - 9:54 [Peer reviews you will see them..] (171:171) (Super)
Codes: [peer reviews.]

Code: Popular methodologies in Banking industry {1-0}

P 9: 08081800 Kabelo.rtf - 9:22 [But with the fact the - my pre..] (29:29) (Super)
Codes: [Popular methodologies in Banking industry]

Code: Problems with case tools {1-0}~

P 3: 07111501 Magda.rtf - 3:25 [So it could help. In my experi..] (150:150) (Super)
Codes: [Problems with case tools]
Memos: [Case tools problem]

So it could help. In my experience through my research, I discovered that sometimes the analysis takes longer than what the people or the developers were used to and then they get discouraged. They should just keep with the process, although it takes longer. In the end you save a lot of time because this is generated automatically (inaudible - noise). My students use this for the last three years - they used Oracle Designer. I saw that with them also, they felt "og, we are busy - we are just busy with the analysis and design, we can't make progress". Then all of a sudden you'll generate the code and then they see.

Code: Pragmatics {0-3}

Code: Problem of Communication {1-0}

P 5: 07121401-Barry Myburgh Corrected.rtf - 5:8 [people operate and communicate..] (27:27) (Super)
Codes: [Problem of Communication]

people operate and communicate very often with a great measure of vagueness and imprecision, and they use words that mean something to one person, that means something different to another person. And that seems to be the reality of how human beings communicate with each other. To expect a computer to be able to cope with that kind of lack of precision in the way that people speak, and the words they use, and the meanings that they attach to those words, I actually think - I don't quite agree with this.

Code: Problem of reusing requirements across organizations {2-0}

P 8: 08072200 and 08072201-Sindiso.rtf - 8:28 [Requirements on their own ...are..] (94:94) (Super)
Codes: [Problem of reusing requirements across organizations]

P 8: 08072200 and 08072201-Sindiso.rtf - 8:29 [Or you could find the frame, t..] (94:94) (Super)
Codes: [Problem of reusing requirements across organizations]

Code: Problem with design model {1-0}~

P 8: 08072200 and 08072201-Sindiso.rtf - 8:10 [So, in essence it bypasses a l..] (59:59) (Super)
Codes: [Problem with design model]

Code: Problem with use of Object Oriented Paradigm {1-0}

P 5: 07121401-Barry Myburgh Corrected.rtf - 5:6 [My problem as I see in industr..] (19:19) (Super)
Codes: [Problem with use of Object Oriented Paradigm]

My problem as I see in industry where I operate, people call things object-oriented when in fact it's just another name for structured modular implementation. And I therefore, you know, I think there's a great lack of - in the broader IT space, there's a great lack of effective utilisation of the object-oriented paradigm.

Code: Problems in Software development {3-0}

P 5: 07121401-Barry Myburgh Corrected.rtf - 5:18 [But of course the collaboratio..] (59:59) (Super)
Codes: [Problems in Software development]

But of course the collaboration is easier said than done because people are not available. The client is running his or her business and that's their first priority and they can't suddenly turn off that part of their life to join your project team and be with you all the time. So it's really a very difficult thing to achieve.

P 5: 07121401-Barry Myburgh Corrected.rtf - 5:28 [The problem with it of course ..] (103:103) (Super)
Codes: [Problems in Software development]

The problem with it of course is it takes time. That's one of the things that we often don't have and therefore we don't always get it right.

P 8: 08072200 and 08072201-Sindiso.rtf - 8:12 [I agree, it's basically the po..] (66:66) (Super)
Codes: [Need a Variety of Development Tools] [Need for a communication medium.] [Problems in Software development]

Code: Problems of not involving the users {1-0}

P 9: 08081800 Kabelo.rtf - 9:44 [But now you find out that he m..] (105:105) (Super)
Codes: [Communication Problem] [Problems of not involving the users]

Code: Problems of using Case tool {1-0}

P 5: 07121401-Barry Myburgh Corrected.rtf - 5:20 [I've got the unfulfilled hope ..] (71:71) (Super)
Codes: [Problems of using Case tool]

I've got the unfulfilled hope that it would really achieve great productivity gains and so on. Unfortunately, when customers are in the kind of craziness of the business reality, where they think that speed is all that counts, then my experience is that case tools don't necessarily help so much. In fact, in some situations the case tools have been seen to be a hindrance because the case tool effectively forces the people doing the work to be more thorough and to

cover all the aspects of the modelling and sophistication of the solution. And there just isn't time to do all of that so people find the tool becomes a problem; they can't use it effectively. And people tend to discard - they end up discarding the use of the tools and really just go through to the minimum level of sort-of version control tools or something like that.

Code: process-orientated paradigm, {1-2}

P 3: 07111501 Magda.rtf - 3:1 [In the process-orientated para..] (25:25) (Super)

Codes: [process-orientated paradigm,]

In the process-orientated paradigm, the philosophical view of the developers of this methodologies is that the process is the most important aspect of the system

Code: process chart {1-2}~

P 2: 07111500 Goede.rtf - 2:11 [process chart] (51:51) (Super)

Codes: [process chart]

process chart

Code: Process Ontology {1-0}

P 1: 07110506 Ernest.rtf - 1:61 [So what I'm saying is that you..] (165:165) (Super)

Codes: [Ontology Knowledge Base] [Process Ontology]

Memos: [ME - 11/13/08 [8]]

So what I'm saying is that you need some formal way where you can have a database that has some schemas of those words, of those concepts that we're talking about. For example, if we talk about a loan, all the financial institutions will have something called a loan. So you can have a loan schema, you find that your Nedbank, your ABSAs and Standard Banks might actually have different terms for your interest - for the interest rate, for whatever, for the borrowing rate or whatever - you know, those technical terms that they use. They might have different terms that actually mean the same thing, so your schema must be able to capture that - ensure that it is the same thing so that when you now check back and re-use it in different institutions you are not really changing much. It will come out there according to their language but you are still using the same thing

Code: Prominent development approaches {1-1}~

P 3: 07111501 Magda.rtf - 3:9 [You mentioned only two of thos..] (43:43) (Super)

Codes: [Prominent development approaches]

Memos: [Prominent development approaches]

You mentioned only two of those approaches and in my research I saw that those two are the mostly used approaches in the industry. You find that there's a movement towards rapid application development but the structured and the object-orientated approach are indeed, I think, the most used approaches.

Code: proof-of-concept {3-1}~

P 1: 07110506 Ernest.rtf - 1:32 [proof-of-concept which was mor..] (79:79) (Super)

Codes: [proof-of-concept]

proof-of-concept which was more practical, where you're coming up with a demo of the system, how the system is going to work. That's basically what really impressed them because they could now start contributing and saying "okay, at this point we want this" or "at this point, no, we don't deliver this kind of reports.

P 2: 07111500 Goede.rtf - 2:12 [A prototype tells you sometime..] (51:51) (Super)
Codes: [proof-of-concept]

A prototype tells you sometimes that - it's empty behind that screen and the proof-of-concept has some functionality and you should have a bit of functionality.

P 2: 07111500 Goede.rtf - 2:19 [that is to do a very good veri..] (63:63) (Super)
Codes: [proof-of-concept]

that is to do a very good verification after the requirements has been gathered. That can be done, once again, with the proof-of-concept.

Code: prototype {2-3}

P 2: 07111500 Goede.rtf - 2:13 [A prototype tells you sometime..] (51:51) (Super)
Codes: [prototype]

A prototype tells you sometimes that - it's empty behind that screen and the proof-of-concept has some functionality and you should have a bit of functionality.

P 3: 07111501 Magda.rtf - 3:15 [You know, we can talk and peop..] (72:72) (Super)
Codes: [prototype]

You know, we can talk and people can come and they can tell you stuff, and then I would use prototypes. And then just very, the prototypes that you can throw away after you've done your requirements. If you re-use your prototypes, that that prototype might become the system and then you don't focus on the requirements. So I would have - my user and I would use prototyping all the way, just to show the user "is this what you want".

Code: Purpose of analysis stage {1-0}

P 9: 08081800 Kabelo.rtf - 9:53 [then we have to move into the ..] (169:169) (Super)
Codes: [Purpose of analysis stage]

Code: quality gate {2-0}~

P 9: 08081800 Kabelo.rtf - 9:30 [with regard to the V model the..] (59:59) (Super)
Codes: [quality gate]

P 9: 08081800 Kabelo.rtf - 9:34 [If ever - those requirements d..] (83:83) (Super)
Codes: [quality gate] [User Involvement]

Code: Quality Measure {1-0}

P 1: 07110506 Ernest.rtf - 1:40 [So the client must be there th..] (107:107) (Super)
Codes: [Quality Measure] [User Involvement]
Memos: [ME - 11/13/08 [1]]

So the client must be there throughout and that is basically for quality check purposes, to ensure that the kind of

requirements that the user has are actually met.

Code: rapid application development... {1-1}~

P 3: 07111501 Magda.rtf - 3:8 [rapid application development...] (39:39) (Super)
Codes: [rapid application development...]

rapid application development.

Code: realistic worldview {1-3}~

P 2: 07111500 Goede.rtf - 2:8 [The realistic worldview says t..] (43:43) (Super)
Codes: [realistic worldview]

The realistic worldview says that everything is agreeable. You and I can decide this is a mouse and this is the function of the mouse. That is the realistic worldview and that is where the problem with the engineers comes in. They follow a realistic worldview. They think they agree on the purpose of this thing.

Code: Reality Centre {1-0}

P 5: 07121401-Barry Myburgh Corrected.rtf - 5:15 [in fact one of my customers a ..] (53:53) (Super)
Codes: [Language Community] [Need for a communication medium.] [Reality Centre]

in fact one of my customers a few years back, they had a centre - a room which was dedicated to sharing information, they called it a reality centre. And that sort-of supports my thinking as well, where the best way to communicate is to use a data projector, you use some kind of tool. I'm a great believer in using mind-mapping as a tool to share thoughts and to achieve consensus. So I would typically project on the screen as the mind-map develops, as I'm engaging with the people that have to be part of the buy-in in the model that is developed, and they participate in the development of the model. So it's very much like in a real JAD session, where you use some tools to develop a model and you involve everyone in visually in this kind of reality centre concept of participating in the development of the model.

Code: regressional testing {1-0}

P 8: 08072200 and 08072201-Sindiso.rtf - 8:52 [regressional testing] (148:148) (Super)
Codes: [regressional testing]

Code: relativistic worldview {1-2}~

P 2: 07111500 Goede.rtf - 2:9 [And if we follow a relativisti..] (45:45) (Super)
Codes: [relativistic worldview]

And if we follow a relativistic worldview, you might have an opinion on this device and I can have a different opinion on this device. And if we then get down to a discussion, we might design a much better mouse than this - if we sit down and think "what is the real purpose".

Code: Requirement for non prescriptive ways of using systems {1-0}~

P 5: 07121401-Barry Myburgh Corrected.rtf - 5:13 [So to my mind, that is the way..] (47:47) (Super)

Codes: [Requirement for non prescriptive ways of using systems]

So to my mind, that is the way that things should be done, that the system is built from the perspective of trying to anticipate any - you know, a whole set of ways in which a user would prefer to actually perform a certain function; and then leave it to the user to choose the route that is most comfortable for him or her.

Code: Requirements for quality software products {1-0}

P 5: 07121401-Barry Myburgh Corrected.rtf - 5:30 [First of all, you have to have..] (111:111) (Super)

Codes: [Requirements for quality software products]

First of all, you have to have enough time to drive out the proper understanding of the requirements of the architecture, and to plan the project against a sufficiently understood reality. If you're in a great hurry and there's no time to do planning, there's no time to develop an architecture and things like that, then you will not get those things right.

Code: Requirements of a Development Tool {4-1}~

P 1: 07110506 Ernest.rtf - 1:33 [Looking at it from that point ..] (89:89) (Super)

Codes: [Requirements of a Development Tool]

Looking at it from that point of view, I would say that the most important parts of a tool that I would expect is that tool is going to be used in reusable systems and adaptable systems and so forth, it should be a tool that is open enough to allow the users to bring in their expertise and skills.

P 1: 07110506 Ernest.rtf - 1:35 [The other thing that I would a..] (91:91) (Super)

Codes: [Requirements of a Development Tool]

Memos: [ME - 11/12/08 [9]]

The other thing that I would also mention about the competence of the tool, is that the tool should not be too much restricted to a particular methodology because, like you said, a methodology can actually change, people can come up with different approaches to that methodology, and change the framework and its parameters. So when that happens, the tool itself must actually be adaptable to that.

P 8: 08072200 and 08072201-Sindiso.rtf - 8:35 [You know, we have a system tha..] (108:108) (Super)

Codes: [Requirements of a Development Tool]

P 8: 08072200 and 08072201-Sindiso.rtf - 8:36 [I mean the system on its own s..] (110:110) (Super)

Codes: [Requirements of a Development Tool] [Requirements of a new development approach]

Code: Requirements of a new development approach {4-0}

P 8: 08072200 and 08072201-Sindiso.rtf - 8:14 [You have a computer system whi..] (68:68) (Super)

Codes: [Requirements of a new development approach]

P 8: 08072200 and 08072201-Sindiso.rtf - 8:25 [first of all I believe we shou..] (88:88) (Super)

Codes: [Need to improve programming languages] [Requirements of a new development approach]

P 8: 08072200 and 08072201-Sindiso.rtf - 8:36 [I mean the system on its own s..] (110:110) (Super)

Codes: [Requirements of a Development Tool] [Requirements of a new development approach]

P 8: 08072200 and 08072201-Sindiso.rtf - 8:37 [So, like I said, it's basicall..] (112:112) (Super)
Codes: [Adaptive systems development approaches] [Requirements of a new development approach] [Romantic Worldview]

Code: Requirements of a software tester {1-0}

P 9: 08081800 Kabelo.rtf - 9:40 [I mean as the software tester,..] (101:101) (Super)
Codes: [Requirements of a software tester]

Code: Requirements Repository {2-2}~

P 2: 07111500 Goede.rtf - 2:24 [Firstly because I think it's m..] (75:75) (Super)
Codes: [Need for a development tool] [Requirements Repository]
Memos: [Requirements Repository]

Firstly because I think it's mostly - let me say, I don't think the end-user will benefit as much as the developer from doing it because you have one version of the truth of the requirement, you have one place - central repository - where you can find the requirements, you can set up checklists to see if you have fulfilled the requirements, the requirements are (nearby??) but then the system must force the developers or the analysts to fill it in.

P 8: 08072200 and 08072201-Sindiso.rtf - 8:33 [So basically I would document ..] (102:102) (Super)
Codes: [Requirements Repository]

Code: Reusability of Requirements {3-0}

P 5: 07121401-Barry Myburgh Corrected.rtf - 5:24 [So I suppose the answer in pri..] (91:91) (Super)
Codes: [Reusability of Requirements]

So I suppose the answer in principle is to ensure that requirements can be re-used, you need to have a sufficiently organised approach to your software engineering so that you can have access to the designs and specifications and architecture of systems that have previously been built. It's to do with configuration management and if your discipline of configuration management isn't effectively implemented, then you just can't achieve that.

P 8: 08072200 and 08072201-Sindiso.rtf - 8:31 [object oriented language is ac..] (96:96) (Super)
Codes: [Reusability of Requirements]

P 8: 08072200 and 08072201-Sindiso.rtf - 8:32 [So, in essence you create obje..] (98:98) (Super)
Codes: [Reusability of Requirements]

Code: Romantic Worldview {1-8}

P 8: 08072200 and 08072201-Sindiso.rtf - 8:37 [So, like I said, it's basicall..] (112:112) (Super)
Codes: [Adaptive systems development approaches] [Requirements of a new development approach] [Romantic Worldview]

Code: schemas {1-0}

P 1: 07110506 Ernest.rtf - 1:60 [So now what I'm trying to say ..] (165:165) (Super)
Codes: [schemas]

So now what I'm trying to say is that you need to come up with some schemas because that language is full of these schemas that you used to, for example, you're trying to come up with a system for the library. In order for you to represent the requirements there, you come up with those schemas that will give you, for example, the - you can

have schema called “loaning a book” or something like that.

Code: scope creep {2-2}~

P 2: 07111500 Goede.rtf - 2:31 [I think scope creep always hap..] (103:103) (Super)

Codes: [scope creep]

I think scope creep always happens but the extent of it depends on your - two things - actually mainly on the quality of your requirements. If your requirements are done well and you involve the right people, you might have a situation where you don't get that much scope through it. And then as - if you are using a methodology where your end-user is involved, then scope should be their problem, not your problem in a sense. But that's only the case if you have a end-user or a - I don't want to use the word end-user, I want to use a word (indiscernible - audio quality) user in data warehousing, business sponsors - somebody higher up in the organisation. Somebody more important than the secretary that's going to type on the system; somebody that has power inside the organisation. I want to have that person involved and he should prioritise all these additional scope requirements, not the programmers.

P 3: 07111501 Magda.rtf - 3:35 [Is spent on 20 percent of the ..] (214:216) (Super)

Codes: [scope creep]

Memos: [Managing scope creep]

Is spent on 20 percent of the requirement. Are they willing to do that? Are they willing to spend 80 percent of the time? If they cut that 20 percent of the requirement, they could save 80 percent of the time you can go onto a next project. So just discuss it with them. Ask them what their philosophy is, I would say, because some people really want that product and they want it in the way they visualise it and they want to add stuff and so on. And if they are willing to pay for that, fine with me, then I will add it because in the end of the day, it's [them] that are paying for the product, not me.

But other companies are very, very, very driven and they want you to produce quick results and then I would show them. Think about it, if you are in your class also, the amount of time you spend with students in your office. It could be one student that takes up all your time in the office. It's the same with this principle. There's a small amount of requirement and it takes 80 percent of your time.

Code: Seeking explanation of concepts {3-0}

P 5: 07121401-Barry Myburgh Corrected.rtf - 5:5 [Maybe you should just explain ..] (13:13) (Super)

Codes: [Seeking explanation of concepts]

Maybe you should just explain a little bit what you mean by your use of the term semantics.

P 5: 07121401-Barry Myburgh Corrected.rtf - 5:7 [What do we mean by natural lan..] (23:23) (Super)

Codes: [Seeking explanantion of concepts]

What do we mean by natural language speakers? I mean, what are we actually talking about there? Do we expect - you know, I don't quite understand the context of this comment. Do you mean in terms of the way that we give them instructions, that we should just be able to communicate our instructions to the computer in natural language? Is that what you mean?

P 5: 07121401-Barry Myburgh Corrected.rtf - 5:10 [You see, now I first have to c..] (31:31) (Super)

Codes: [Seeking explanantion of concepts]

You see, now I first have to check the meaning of these words now. Formal, informal and the technical part, are you

meaning by the technical part, the hardware, the actual physical -

Code: Semantics {0-1}

Code: So you need to try to anticipa.. {1-0}~

P 5: 07121401-Barry Myburgh Corrected.rtf - 5:14 [So you need to try to anticipa..] (49:49) (Super)

Codes: [So you need to try to anticipa..]

So you need to try to anticipate all the different ways that people might want to perform a function and build a system to cater for them.

Code: Soft issues {2-2}~

P 1: 07110506 Ernest.rtf - 1:16 [soft issues] (41:41) (Super)

Codes: [Soft issues]

soft issues

P 1: 07110506 Ernest.rtf - 1:52 [I would say [that] if an indiv..] (141:141) (Super)

Codes: [Soft issues] [Soft Systems Methodology]

I would say [that] if an individual wants to work in that area, they'll have to look into a lot of other areas that may not necessarily be in IT but others are IT. For example going into psychology, going into organisational behaviour. Those are areas that are very, very interesting and there are books that people have written that bring the softer issues of IT and the technical issues together and so forth.

Code: Soft Systems Methodology {2-3}

P 1: 07110506 Ernest.rtf - 1:52 [I would say [that] if an indiv..] (141:141) (Super)

Codes: [Soft issues] [Soft Systems Methodology]

I would say [that] if an individual wants to work in that area, they'll have to look into a lot of other areas that may not necessarily be in IT but others are IT. For example going into psychology, going into organisational behaviour. Those are areas that are very, very interesting and there are books that people have written that bring the softer issues of IT and the technical issues together and so forth.

P 3: 07111501 Magda.rtf - 3:6 [The other paradigm would be - ..] (29:29) (Super)

Codes: [Soft Systems Methodology]

The other paradigm would be - I would call holistic paradigm where they focus on the system - the systems thinking. I don't know if you are familiar with the term systems thinking?

Code: Software Crisis {1-0}

P 5: 07121401-Barry Myburgh Corrected.rtf - 5:29 [The bottom line is that invari..] (111:111) (Super)

Codes: [Software Crisis]

The bottom line is that invariably we don't ensure those things. Okay, we most often don't get it right. So if you are in a situation where you have to ensure that all of those things are achieved, well there's a few key things that you

have to be able to do.

Code: software development metaphor {1-2}~

P 1: 07110506 Ernest.rtf - 1:13 [software development metaphor] (33:33) (Super)
Codes: [software development metaphor]

software development metaphor

Code: Software development methodology requirements {1-1}

P 2: 07111500 Goede.rtf - 2:32 [I think again, once again the ..] (109:109) (Super)
Codes: [Software development methodology requirements]

I think again, once again the human factor is much more important than previously. We should allow for methodologies to really get us to ask the right questions for the people on how they use the system, and what their expectations are. And we should go from a relativistic viewpoint, that we understand that different people will have different views on these things. The more people you ask the better your understanding. The more different answers you get to the same question, you will enrich understanding and not hinder your understanding.

Code: software engineering environme.. {2-2}~

P 3: 07111501 Magda.rtf - 3:23 [There are those types of tools..] (148:148) (Super)
Codes: [software engineering environme..]

There are those types of tools. Do you call it a tool? I would call it - sometimes I call it a software engineering environment. Just to make the difference clear, a tool - I would say a tool would be something that helps you draw a dataflow diagram. But what you described there I would call a software engineering environment. Ja, for example, one example of those is Oracle Designer. I don't know if you know the thing. What you do there, you start with your analysis, you go through to your design, and you do both analysis and design of processes and of your data. You mention the data there. Then in the end, you just generate - automatically generate all the code. If you do it correctly, it's fabulous, it could work wonders but there's a very big danger, you should be careful with your analysis. If you do your analysis wrong, then you can solve the problem but it's the wrong answer to the initial problem

P 5: 07121401-Barry Myburgh Corrected.rtf - 5:19 [If the customer that you are b..] (71:71) (Super)
Codes: [Case Tool] [software engineering environme..]

If the customer that you are busy working for is the kind of customer who understands the importance of giving you enough time to do the job properly, then the use of case tools can greatly improve the software development process.

Code: software evolution {3-2}

P 1: 07110506 Ernest.rtf - 1:47 [software evolution] (129:129) (Super)
Codes: [software evolution]

software evolution

P 1: 07110506 Ernest.rtf - 1:48 [So his concept is that if you ..] (131:131) (Super)
Codes: [software evolution]

So his concept is that if you go electrical engineering - a field of electrical engineering called “control engineering” - you’ll find that in order to stabilise a system, they put all systems in black boxes, especially the processes - industrial processes. They’ll put it in a black box and say “a system is an input, an output”. And then in order to stabilise that system, you are saying the output must be taped as either a part of the output or the entire output. You’ll tape it and fit it back into the input, so there must be a mixer there - some function which you represent by a summation sign to say “we are summing the input and part of the feedback”. But in so doing you are regulating the output so that the output does not spiral exponentially. Right, and that concept of spiralling exponentially you’ll find it - let’s say in speakers - if you put a mic next to the speaker, then you have negative feedback. So you are trying to control the feedback there.

P 1: 07110506 Ernest.rtf - 1:49 [So in software evolution now t..] (133:133) (Super)
Codes: [software evolution]

So in software evolution now the Lehman concept actually says that the moment you deploy the system, and the user starts using the system, they’ll tell you some things that they don’t like about the system and whatever - things that need to be changed about the system. So that should be considered as feedback from the system that you have delivered. So take that feedback and feed it together with the input, to control the way your system should actually work. So that’s the concept. It’s trying to use that control engineering approach to actually regulate the way we develop our systems.

Code: Software Maintainability {1-0}

P 1: 07110506 Ernest.rtf - 1:46 [use software patterns and thos..] (121:121) (Super)
Codes: [Software Maintainability]

use software patterns and those will actually assist in terms of the maintainability of the system. But object-orientation is the heart of the maintainable systems because you have a lot of information hiding and whatever, and modularisation, each class has its own internals that can not be touched by any other class. So that when you do your maintenance and checks, the methods inside a class, it doesn’t affect all the other classes

Code: Software migration issues and problems {1-0}~

P 8: 08072200 and 08072201-Sindiso.rtf - 8:6 [And believe me, the amount of ..] (40:40) (Super)
Codes: [Software migration issues and problems]

Code: software patterns. {1-3}

P 1: 07110506 Ernest.rtf - 1:45 [software patterns.] (117:117) (Super)
Codes: [software patterns.]

software patterns.

Code: software tester {1-0}

P 9: 08081800 Kabelo.rtf - 9:39 [software tester] (101:101) (Super)
Codes: [software tester]

Code: star schemas {2-1}

P 2: 07111500 Goede.rtf - 2:15 [star schemas] (55:55) (Super)
Codes: [star schemas]

star schemas

P 2: 07111500 Goede.rtf - 2:16 [they are the basic design. The..] (59:59) (Super)
Codes: [star schemas]

they are the basic design. They replace the ERD. They replace the ERD but they are designed in such a way that I can teach you within three seconds how a star schema works but I won't even attempt to teach anybody an ERD.

Code: Studying the system {1-1}~

P 3: 07111501 Magda.rtf - 3:21 [It could be with experience bu..] (120:120) (Super)
Codes: [Studying the system]

It could be with experience but I think what you mentioned here is the big danger. You shouldn't do that; you shouldn't think because it looks the same, that it is the same. You should try and figure out - there's always differences. You will know.

Code: Super users {1-0}~

P 7: 08072102 Edwin.rtf - 7:10 [Whereas normally the super-use..] (57:57) (Super)
Codes: [Super users]

Code: System Developer {4-1}~

P 1: 07110506 Ernest.rtf - 1:24 [In the approach that I've stud..] (61:61) (Super)
Codes: [System Developer]

In the approach that I've studied in detail and also used, you'll find that the gap between the analyst and the programmer is actually closed by coming up with somebody who's called a developer, who actually elicits the requirements directly from the user. And that user, in the methods that I'm actually talking about, becomes part of the development team which means that those details - those metaphors that I was talking about - the language that is used by the team becomes familiar to that user. And the problem - the architectural problems where the systems analyst has some design diagrams and whatever that may not be clear to the programmer ?

P 2: 07111500 Goede.rtf - 2:23 [Now I've seen a number of orga..] (71:71) (Super)
Codes: [System Developer]
Memos: [System Developer]

Now I've seen a number of organisations employing people that has a little bit of technical skill a lot of business skills to serve as the interface between the technical people and the users.

P 8: 08072200 and 08072201-Sindiso.rtf - 8:40 [And I personally believe an an..] (124:124) (Super)
Codes: [System Developer]

P 8: 08072200 and 08072201-Sindiso.rtf - 8:42 [So I would want a situation wh..] (126:126) (Super)
Codes: [System Developer]

Code: System testing {3-0}~

P 8: 08072200 and 08072201-Sindiso.rtf - 8:49 [Ok, there are different method..] (148:148) (Super)
Codes: [System testing]

P 9: 08081800 Kabelo.rtf - 9:6 [system testing] (9:9) (Super)
Codes: [System testing]

P 9: 08081800 Kabelo.rtf - 9:41 [Meaning when I'm doing system ..] (103:103) (Super)
Codes: [System testing]

Code: System testing concept {1-0}

P 9: 08081800 Kabelo.rtf - 9:12 [From there the analyst, meanin..] (9:9) (Super)
Codes: [System testing concept]

Code: systems approach {1-3}~

P 2: 07111500 Goede.rtf - 2:1 [systems approach] (19:19) (Super)
Codes: [systems approach]

systems approach

Code: tacit knowledge {1-1}~

P 1: 07110506 Ernest.rtf - 1:2 [tacit knowledge] (23:23) (Super)
Codes: [tacit knowledge]

tacit knowledge

Code: Test Analyst duties {1-0}

P 9: 08081800 Kabelo.rtf - 9:1 [Meaning I run my test scripts ..] (3:3) (Super)
Codes: [Test Analyst duties]

Code: test design techniques {1-0}

P 9: 08081800 Kabelo.rtf - 9:7 [test design techniques] (9:9) (Super)
Codes: [test design techniques]

Code: Test Director {1-0}~

P 8: 08072200 and 08072201-Sindiso.rtf - 8:18 [There's a system called Test D..] (74:74) (Super)
Codes: [Communication Method] [Test Director]

Code: Test Driven Development {1-0}

P 7: 08072102 Edwin.rtf - 7:2 [Test Driven Development] (35:35) (Super)
Codes: [Test Driven Development]

Code: timebox approach {1-0}

P 5: 07121401-Barry Myburgh Corrected.rtf - 5:35 [In a crafted quality or agile ..] (119:119) (Super)
Codes: [Adaptive systems development approaches] [timebox approach]

In a crafted quality or agile environment, I certainly don't make promises about things like that, and I say "give me a timebox, let's set some goals and objectives for that timebox and we will do the best we can". And at the end of the timebox we will review what we have achieved. So I can certainly ensure that the budget is adhered to because

the timebox gives me a certain budget. I can achieve on time because the timebox is limited, say to six weeks, two months, three months, whatever; but I can't guarantee that I what I build will be easy to modify, fault free, and that the user will be satisfied. That we will see - on reflection see to what extent we have achieved that. If the customer says "you've achieved enough to gain my confidence" then we get another timebox to improve those things.

Code: timebox approach" {1-0}~

P 5: 07121401-Barry Myburgh Corrected.rtf - 5:34 [timebox approach"] (117:117) (Super)

Codes: [timebox approach"]

timebox approach"

Code: traditional approach {3-3}~

P 1: 07110506 Ernest.rtf - 1:23 [traditional approach, where yo..] (61:61) (Super)

Codes: [traditional approach]

traditional approach, where you have a systems analyst, you have a programmer, and then you have got the business guy, the user.

P 2: 07111500 Goede.rtf - 2:2 [I'm very much against the trad..] (19:19) (Super)

Codes: [traditional approach]

Memos: [ME - 01/12/09]

I'm very much against the traditional paradigm because I believe it leads to a case where you have a system that conforms to the requirements and not to the user's needs.

P 3: 07111501 Magda.rtf - 3:3 [In the process-orientated para..] (25:25) (Super)

Codes: [traditional approach]

Memos: [Structured development approach]

In the process-orientated paradigm, the philosophical view of the developers of this methodologies is that the process is the most important aspect of the system. And the data-orientated paradigm is where they say [that] data stays constant, so we should rather focus on the data instead of the process, it would make our lives much easier.

Code: Translation Medium Requirement {1-1}~

P 1: 07110506 Ernest.rtf - 1:22 [So you have the business that ..] (57:57) (Super)

Codes: [Development Problem] [Translation Medium Requirement]

So you have the business that side, you have got the interface which the user is using, and you have the technical details of the system itself. So those are three different levels and you find that at those three different levels the user understands the business side and the interface, but does not really understand the technical details but the developer who is the one who has the technical details. So you find that that way now, those requirements will always be presented in the way that the technical guy understands which puts the user at a disadvantage.

Code: UML {4-2}~

P 1: 07110506 Ernest.rtf - 1:18 [So it's more of the - when you..] (47:47) (Super)
Codes: [UML]
Memos: [ME - 11/12/08 [5]]

So it's more of the - when you look at an object, it gives a better understanding of the system or whatever you're dealing with but I wouldn't call it semantics.

P 1: 07110506 Ernest.rtf - 1:19 [I think it's lighter than sema..] (49:49) (Super)
Codes: [UML]

I think it's lighter than semantics. That's my bottom-line, that it doesn't give the details that semantics would give. It leaves you at a general level again and that is the advantage of object-oriented programme and the development (inaudible - noise) technology. Because with that general class you can actually inherit and start reusing the system and the like without really going into the semantic details. So I would say there's a level of understanding that has been increased, as opposed to structured programming but they still need to go deeper which would now become the semantic

P 8: 08072200 and 08072201-Sindiso.rtf - 8:4 [I mean UML it's being used as ..] (34:34) (Super)
Codes: [UML]

P 9: 08081800 Kabelo.rtf - 9:37 [Okay we use your normal UML to..] (87:87) (Super)
Codes: [UML]

Code: unit testing {1-0}

P 8: 08072200 and 08072201-Sindiso.rtf - 8:56 [unit testing] (148:148) (Super)
Codes: [unit testing]

Code: use case diagrams {2-2}~

P 2: 07111500 Goede.rtf - 2:5 [use case diagrams] (23:23) (Super)
Codes: [use case diagrams]
Memos: [ME - 01/12/09 [2]]

use case diagrams

P 2: 07111500 Goede.rtf - 2:7 [use case diagrams. You can use..] (39:39) (Super)
Codes: [use case diagrams]
Memos: [ME - 01/12/09 [4]]

use case diagrams. You can use UML quite well. But one should not stick to the rigidness of such techniques or the prescriptiveness; one should be able to add your own things, like to be flexible. And nowadays (inaudible - noise) it's too prescribed. We don't have enough space on those diagrams to write down what its purpose originally was, and that is to capture non-technical information.

Code: User acceptance testing {3-0}

P 8: 08072200 and 08072201-Sindiso.rtf - 8:16 [It is one of the processes act..] (74:74) (Super)
Codes: [User acceptance testing]

P 8: 08072200 and 08072201-Sindiso.rtf - 8:19 [So basically it's the UAT proc..] (76:76) (Super)
Codes: [User acceptance testing]

P 9: 08081800 Kabelo.rtf - 9:13 [hey'll have to do user accepta..] (11:11) (Super)
Codes: [User acceptance testing]

Code: user design approach {1-0}~

P 9: 08081800 Kabelo.rtf - 9:15 [But because we follow the user..] (15:15) (Super)
Codes: [user design approach]

Code: user interface {2-0}

P 2: 07111500 Goede.rtf - 2:6 [user interface] (27:27) (Super)
Codes: [user interface]
Memos: [ME - 01/12/09 [3]]

user interface

P 3: 07111501 Magda.rtf - 3:22 [that user interface sketches th..] (144:144) (Super)
Codes: [user interface]

that user interface sketches that you talk about could be (inaudible - noise) ja. Ja, I do agree with that. I'm sure. If you show them this prototypes or these interface sketches, like you call it, that they would understand it much better. To them the system is the interface,

Code: User Interface Problem {1-0}~

P 1: 07110506 Ernest.rtf - 1:20 [But now the major disadvantage..] (55:55) (Super)
Codes: [User Interface Problem]

But now the major disadvantage that I would bring forward is that it sort-of clouds the understanding of the user, especially when it comes to issues like maintenance where you have new requirements that are supposed to be implemented, because the user does not know how the existing methods are actually working, the general problems would be to try and elicit the correct requirements from that individual.

Code: User Involvement {10-2}~

P 1: 07110506 Ernest.rtf - 1:28 [So sitting together there is a..] (67:67) (Super)
Codes: [Language Community] [User Involvement]

So sitting together there is a very important aspect or practice of eliciting the requirements or systems development. You are saying that these three guys should be together. You don't want the systems analyst to meet some guys there, get some documents signed and then go and shift that information into the programme, but you want these three levels to actually sit together. The user must be the centre of all those things.

P 1: 07110506 Ernest.rtf - 1:40 [So the client must be there th..] (107:107) (Super)
Codes: [Quality Measure] [User Involvement]
Memos: [ME - 11/13/08 [1]]

So the client must be there throughout and that is basically for quality check purposes, to ensure that the kind of requirements that the user has are actually met.

P 2: 07111500 Goede.rtf - 2:17 [The other way is to incorporat..] (63:63) (Super)
Codes: [User Involvement]

The other way is to incorporate the clients right through the process and not only in the requirements phase - but to talk to client's right through.

P 2: 07111500 Goede.rtf - 2:34 [Involving your users in all th..] (131:131) (Super)

Codes: [User Involvement]

Memos: [Software application domain]

Involving your users in all the stages, so that we don't spend a lot of time on things that's not important. That's I think - and I look at my students and see what they do wrong. They spend hours and hours programming stuff that the end-user are not interested in - in aesthetics and things. Aesthetics of specific functions or functionality the user is normally not interested in. Their priorities are incorrect.

P 5: 07121401-Barry Myburgh Corrected.rtf - 5:17 [Obviously the most effective w..] (57:57) (Super)

Codes: [Collaborative Approach] [User Involvement]

Obviously the most effective way of trying to ensure that is if we have the client participate in the activities of the project. In other words, a collaborative approach where the client is really a part of the team.

P 7: 08072102 Edwin.rtf - 7:6 [So if you interact with the us..] (49:49) (Super)

Codes: [User Involvement]

DATA SOURCE UNACCESSIBLE!: C:\Documents and Settings\user\My Documents\Working Folder 05 11 2008\PhD Current\VOICE\Transcribed Interviews\08072102 Edwin.rtf

P 9: 08081800 Kabelo.rtf - 9:23 [So the user involvement is alw..] (43:43) (Super)

Codes: [User Involvement]

P 9: 08081800 Kabelo.rtf - 9:29 [If ever, in terms of the walkt..] (53:53) (Super)

Codes: [User Involvement]

P 9: 08081800 Kabelo.rtf - 9:34 [If ever - those requirements d..] (83:83) (Super)

Codes: [quality gate] [User Involvement]

P 9: 08081800 Kabelo.rtf - 9:42 [Because obviously I'll be test..] (103:103) (Super)

Codes: [User Involvement]

Code: user stories {3-3}~

P 1: 07110506 Ernest.rtf - 1:26 [That's an approach that is ver..] (65:65) (Super)

Codes: [user stories]

That's an approach that is very, very powerful that a lot of object-oriented programmers actually use today, especially in the agile world. So with user stories, you'll ask the user to give you a different story of what functionality they would like to be implemented. So it's not a use case, it's not as detailed as a use case but you give them some 6 x 6 inch cards and ask them to write a little functionality that they want implemented. So you're breaking down the requirements into those small user stories. Then you take those user stories now and ask the guy who is going to develop to now try and estimate how long it's going to take to actually develop that. Then you are coming up with some design already as you do that.

P 3: 07111501 Magda.rtf - 3:33 [What they do, they get the req..] (192:192) (Super)

Codes: [user stories]

What they do, they get the requirement on user cards. You know the user stories? I don't ?

P 3: 07111501 Magda.rtf - 3:34 [Normally they're the small car..] (196:196) (Super)
Codes: [user stories]

Normally they're the small cards like - something like this size - and then the users would write their requirements here of the system. And what they do, in the end they collect all the cards and they put a price tag on it. The price in terms of the time it would take and the physical resources they would need and (inaudible - speaking softly) the money. And then they tell the users "what would you like us to do in the following three weeks". And then the users would collect some of those and then it becomes a chunk. So they work only on those requirements and they implement it,

Code: Users' perception of system {1-0}

P 8: 08072200 and 08072201-Sindiso.rtf - 8:17 [What we do is whenever we deve..] (74:74) (Super)
Codes: [Users' perception os system]

Code: V-model {2-0}~

P 9: 08081800 Kabelo.rtf - 9:2 [Basically, ... in terms of metho..] (7:7) (Super)
Codes: [V-model]

P 9: 08081800 Kabelo.rtf - 9:21 [waterfall or the model which V..] (29:29) (Super)
Codes: [V-model]

Code: walk-throughs {1-0}

P 9: 08081800 Kabelo.rtf - 9:26 [walk-throughs] (51:51) (Super)
Codes: [walk-throughs]

Code: waterfall approach {2-0}

P 5: 07121401-Barry Myburgh Corrected.rtf - 5:1 [For situations where it is pos..] (1:1) (Super)
Codes: [waterfall approach]
Memos: [Suitability of Waterfall approach]

For situations where it is possible to clearly define what is required, and where there is enough time to do effective planning, I would follow a pretty conventional waterfall approach.

P 5: 07121401-Barry Myburgh Corrected.rtf - 5:31 [The one I call "controlled qua..] (113:113) (Super)
Codes: [controlled quality"] [waterfall approach]

The one I call "controlled quality"; the other one I call "crafted quality". Now crafted quality is very much what most people call agile (inaudible - cross-talking) okay; and controlled quality is very much the kind-of conventional waterfall lifecycle approach.

Code: What is the research project addressing? {1-0}

P 9: 08081800 Kabelo.rtf - 9:52 [So, is it more like, in terms ..] (159:159) (Super)
Codes: [What is the research project addressing?]

Code: whiteboards and flyers {2-2}

P 1: 07110506 Ernest.rtf - 1:11 [whiteboards and flyers] (31:31) (Super)

Codes: [whiteboards and flyers]
Memos: [ME - 11/12/08 [2]]

whiteboards and flyers

P 1: 07110506 Ernest.rtf - 1:34 [printable whiteboards] (89:89) (Super)
Codes: [whiteboards and flyers]

printable whiteboards

Code: workshop {1-0}~

P 7: 08072102 Edwin.rtf - 7:8 [Organise a workshop and then y..] (57:57) (Super)
Codes: [workshop]

Code: Workshops for communicating system functionality with users {2-1}

P 1: 07110506 Ernest.rtf - 1:37 [So normally a good number of m..] (101:101) (Super)
Codes: [Workshops for communicating system functionality with users]

So normally a good number of methodologies that I've studied would have a lot of these workshops where you actually go into the details of how the functionalities of the system are supposed to be. So you find that sometimes you have these workshops and people still remain without the correct understanding.

P 9: 08081800 Kabelo.rtf - 9:24 [Jad sessions, walk-throughs me..] (51:51) (Super)
Codes: [Workshops for communicating system functionality with users]

Appendix L: Full List of codes and code families for all the interviews (on CD)

HU: An Ontological Approach to Software Development
File: [C:\Documents and Settings\NMavetera\My Docum...An Ontological Approach to Software Development.hpr5]
Edited by: Super
Date/Time: 10/09/03 05:33:44 PM

Code Family: Adaptive Products Development Technique

Created: 08/11/13 02:52:29 PM (Super)

Codes (10): [adaptive evolutionary approach..] [Adaptive Software Products] [Adaptive software products.] [Adaptive Systems] [Adaptive systems development approaches] [configurable systems] [heuristic systems] [Humanist Requirement for Software Development] [intuitive system] [Need to adapt systems to organizational requirements]

Quotation(s): 17

Code Family: Communication Method

Created: 08/11/12 02:29:38 PM (Super)

Comment:

This a method needed or used during software development. The method is supposed to improve the way stakeholders share and understand concepts used in the organisation. The purpose of the method is to develop a language community.

Codes (5): [brainstorming forum] [business analysts] [Discussion Forum] [User Involvement] [user stories]

Quotation(s): 19

Code Family: Communication Technique

Created: 08/11/12 02:29:16 PM (Super)

Codes (13): [brainstorming forum] [Concept negotiation Technique] [Discussion Forum] [ERDs] [Pair Programming] [proof-of-concept] [prototype] [software development metaphor] [UML] [use case diagrams] [user interface] [user stories] [whiteboards and flyers]

Quotation(s): 27

Code Family: Communication Tool

Created: 08/11/12 02:29:52 PM (Super)

Codes (5): [process chart] [UML] [use case diagrams] [user stories] [whiteboards and flyers]

Quotation(s): 12

Code Family: Contextual Issues

Created: 08/11/12 02:33:17 PM (Super)

Codes (18): [brainstorming forum] [business analysts] [Capturing Pragmatic Knowledge] [Developer Understanding of Business Model] [Discussion Forum] [Knowledge Sharing] [language database] [Need for a communication medium.] [Ontology Knowledge Base] [Pair Programming] [Process Ontology] [relativistic worldview] [Soft issues] [Soft Systems Methodology] [Studying the system] [tacit knowledge] [User Involvement] [user stories]

Quotation(s): 38

Code Family: Development Approach

Created: 08/11/12 02:25:59 PM (Super)

Codes (10): [agile methodologies] [contingency approach,] [Incremental development] [New Development Approach] [object-oriented approach,] [Prominent development approaches] [rapid application development...] [Soft issues] [systems approach] [traditional approach]

Quotation(s): 24

Code Family: Development Issues

Created: 08/11/12 02:32:36 PM (Super)
Codes (0):
Quotation(s): 0

Code Family: Development Method
Created: 08/11/12 02:26:13 PM (Super)
Codes (10): [component development] [data-orientated paradigm] [Extreme programming] [Incremental development] [information engineering] [interactive development] [Pair Programming] [process-orientated paradigm.] [rapid application development...] [Soft Systems Methodology]
Quotation(s): 14

Code Family: Development Problem
Created: 08/11/12 02:28:05 PM (Super)
Codes (30): [Ambiguity on methodology that captures semantics] [Capturing Human Behavior] [Capturing Pragmatic Knowledge] [Capturing Semantics Issue] [Communication Problem] [Communication problem between the conceptual and physical design.] [Concept negotiation Technique] [Developer Understanding of Business Model] [Development Problem] [Evolving Product development Technique] [Humanist Requirement for Software Development] [Knowledge reuse] [Knowledge Sharing] [Language Community] [language database] [Language Limitations] [Language Requirement] [Need for a communication medium.] [Porblems with case tools] [realistic worldview] [relativistic worldview] [Requirements Repository] [scope creep] [Soft issues] [software evolution] [tacit knowledge] [traditional approach] [Translation Medium Requirement] [User Interface Problem] [User Involvement]
Quotation(s): 75

Code Family: Development Technique
Created: 08/11/12 02:26:34 PM (Super)
Codes (17): [brainstorming forum] [Concept negotiation Technique] [data-orientated paradigm] [Discussion Forum] [ERDs] [Evolving Product development Technique] [Extreme programming] [Language Community] [Oracle Designer] [Pair Programming] [software development metaphor] [software patterns.] [UML] [use case diagrams] [User Involvement] [user stories] [Workshops for communicating system functionality with users]
Quotation(s): 43

Code Family: Development Tool
Created: 08/11/12 02:27:05 PM (Super)
Codes (4): [language database] [Ontology Knowledge Base] [UML] [whiteboards and flyers]
Quotation(s): 8

Code Family: Discussion Tools
Created: 08/11/12 03:17:03 PM (Super)
Codes (1): [brainstorming forum]
Quotation(s): 1

Code Family: Evolvable Products Developmet Technique
Created: 08/11/13 02:51:37 PM (Super)
Codes (5): [component development] [Evolvable Software Products Concept] [Evolving Product development Technique] [Incremental development] [software evolution]
Quotation(s): 13

Code Family: Interface Issues
Created: 08/11/12 02:32:20 PM (Super)
Codes (3): [User Interface Problem] [User Involvement] [user stories]
Quotation(s): 14

Code Family: Knowledge Reuse



Created: 08/11/13 05:01:46 PM (Super)
Codes (2): [Knowledge reuse] [Requirements Repository]
Quotation(s): 3

Code Family: Ontology Requirement
Created: 08/11/13 05:28:55 PM (Super)
Codes (0):
Quotation(s): 0

Code Family: Pragmatics
Created: 08/11/12 02:31:30 PM (Super)
Codes (1): [tacit knowledge]
Quotation(s): 1

Code Family: Semantics
Created: 08/11/12 02:31:39 PM (Super)
Codes (6): [Ambiguity on methodology that captures semantics] [Capturing Semantics Issue] [Concept negotiation Technique] [Humanist Requirement for Software Development] [Ontology Knowledge Base] [Process Ontology]
Quotation(s): 11

Code Family: Syntactics
Created: 08/11/12 02:31:49 PM (Super)
Codes (0):
Quotation(s): 0

Code Family: User Involvement
Created: 08/11/12 02:30:19 PM (Super)
Codes (4): [brainstorming forum] [Discussion Forum] [User Involvement] [user stories]
Quotation(s): 16

Appendix M: Full list of incidents and memos for all the interviews (on CD)

HU: An Ontological Approach to Software Development
File: [C:\Documents and Settings\NMavetera\My Docum...\An Ontological Approach to Software Development.hpr5]
Edited by: Super
Date/Time: 10/09/03 05:50:13 PM

MEMO: Adaptive systems (0 Quotations) (Super, 09/04/23 06:25:14 PM)

No codes
No memos
Type: Memo

Adaptive systems consider the context of the task and change accordingly to suit the environment.

MEMO: Agile Approach (0 Quotations) (Super, 09/04/23 11:09:42 AM)

No codes
No memos
Type: Memo

Agile approach is a methodology suitable for developing systems fast and where time is a very limited resource.

MEMO: Business analyst (0 Quotations) (Super, 09/04/28 07:41:28 PM)

No codes
No memos
Type: Memo

This combination of duties may be used to maintain the business model of the system throughout the developmental stages.

MEMO: Case Tools (0 Quotations) (Super, 09/04/23 06:16:09 PM)

No codes
No memos
Type: Memo

Case tools can both be useful or a hindrance during software development. Time factor, the prescriptiveness of the tools do not help much when it comes to developing systems on time. However, the case tools as an example of a software engineering environment can increase communication during software development and as well as the quality of the end product.

MEMO: Case tools problem (1 Quotation) (Super, 09/01/12 08:24:20 PM)

P 3: 07111501 Magda.rtf:
150-150

No codes
No memos
Type: Memo

Case tools take more time during analysis, the fact being that they do not reuse previous requirements. An ontology driven tool may reuse this knowledge.

P 3: 07111501 Magda.rtf - 3:25 [So it could help. In my experi..] (150:150) (Super)

Codes: [Problems with case tools]
Memos: [Case tools problem]

So it could help. In my experience through my research, I discovered that sometimes the analysis takes longer than what the people or the developers were used to and then they get discouraged. They should just keep with the process, although it takes longer. In the end you save a lot of time because this is generated automatically (inaudible - noise). My students use this for the last three years - they used Oracle Designer. I saw that with them also, they felt "e.g. we are busy - we are just busy with the analysis and design, we can't make progress". Then all of a sudden you'll generate the code and then they see.

MEMO: Communication Problem (0 Quotations) (Super, 09/04/23 11:32:57 AM)

No codes
No memos
Type: Memo

Computers cannot understand language as used by humans. This is because of the tacit nature of most of meanings associated with words.

MEMO: Communication requirements (0 Quotations) (Super, 09/04/28 07:48:29 PM)

No codes
No memos
Type: Memo

A methodology must encompass communication methods that facilitate the development of a shared understanding in the development process. This supports the need of a knowledge base, such as an ontology knowledge base.

MEMO: Communication with Stakeholders (0 Quotations) (Super, 09/04/23 12:57:30 PM)

No codes
No memos
Type: Memo

During the formatory stages of software development, visual tools such as mind maps, projectors can be used during JAD sessions as communication methods. These may become difficult to use when the software has been developed and is ready to be packaged.

MEMO: Document of inputs (0 Quotations) (Super, 09/04/28 02:04:49 PM)

No codes
No memos
Type: Memo

The ontology knowledge base can be used instead of a document of inputs. The knowledge base is crafted from the analysis and contains the analysis, design, implementation models.

MEMO: Expropriation of Meaning (0 Quotations) (Super, 09/04/23 11:15:55 AM)

No codes

No memos

Type: Memo

Many respondents did not understand the concepts used by the researcher. Instead, they would ask the researcher to explain the context with in which the concepts were being used in the research.

MEMO: Formal Part (0 Quotations) (Super, 09/04/23 11:47:39 AM)

No codes

No memos

Type: Memo

System developers more often concentrate on the formal part of the system when implementing the system. They do not consider the informal part, a part that captures the tacit knowledge inherent in organizations as an important aspect during development.

MEMO: Knowledge Base (0 Quotations) (Super, 09/04/23 06:38:35 PM)

No codes

No memos

Type: Memo

There is very little emphasis on the reusability of knowledge in practice. This is supported by the fact that knowledge bases for previously acquired knowledge are nonexistent in many software development organizations. In the contrary however, since knowledge reuse improves the quality and speed at which software can be developed, practitioners should be able to create knowledge bases of their past experiences.

MEMO: Lack of trust (1 Quotation) (Super, 09/01/12 07:54:19 PM)

P 3: 07111501 Magda.rtf:

72-72

No codes

No memos

Type: Memo

There is a lot of mistrust between developers and users. This lack of trust leads to poor communication, resulting in poor requirements gathering.

P 3: 07111501 Magda.rtf - 3:16 [I don't know about you Nehemiah..] (72:72) (Super)

Codes: [Development Problem]

Memos: [Lack of trust]

I don't know about you Nehemiah but - there's sort-of a hostility between users and system developers. The users, they don't trust the developers and the developers talk down on the user. Okay, and that's something that's still alive

and well and that bothers me. We are partners' —

MEMO: Language barrier (1 Quotation) (Super, 09/01/12 08:01:20 PM)

P 3: 07111501 Magda.rtf:

84-84

No codes

No memos

Type: Memo

Plain language, understood by all the stakeholders should be used during communication when doing requirements gathering. It is also important to include a business analyst, a person with business orientation to do the analysis. This is like capturing the domain and business model of the system. Then the systems analysts can be incorporated to capture the system requirements.

P 3: 07111501 Magda.rtf - 3:19 [Plain common language; not tec..] (84:84) (Super)

No codes

Memos: [Language barrier]

Plain common language; not technical jargon. Sometimes we don't even - we are not familiar with our own language, the terms we use, the slang that we use. So we should really, really use just plain common language at the level that the user can understand. I think sometimes - you know, nowadays people are very interested in business analysts or business analytics or whatever they call it, where you have business people talking to your users. But I would go further, I wouldn't just have a business person there, I would like a technical person with business knowledge to do this, to do the communication. Ja. Do you have such - I think you have a B.com degree at Mafikeng?

MEMO: Managing scope creep (1 Quotation) (Super, 09/01/12 08:47:38 PM)

P 3: 07111501 Magda.rtf:

214-216

No codes

No memos

Type: Memo

Since user requirements gathering take 80% of the development time, it is important to get a development tool that speeds up the process. Without that, developers risk rushing the requirements gathering process and implementing a wrong system.

P 3: 07111501 Magda.rtf - 3:35 [Is spent on 20 percent of the ..] (214:216) (Super)

Codes: [scope creep]

Memos: [Managing scope creep]

Is spent on 20 percent of the requirement. Are they willing to do that? Are they willing to spend 80 percent of the time? If they cut that 20 percent of the requirement, they could save 80 percent of the time you can go onto a next project. So just discuss it with them. Ask them what their philosophy is, I would say, because some people really want that product and they want it in the way they visualise it and they want to add stuff and so on. And if they are willing to pay for that, fine with me, then I will add it because in the end of the day, it's [them] that are paying for the product, not me.

But other companies are very, very, very driven and they want you to produce quick results and then I would show them. Think about it, if you are in your class also, the amount of time you spend with students

in your office. It could be one student that takes up all your time in the office. It's the same with this principle. There's a small amount of requirement and it takes 80 percent of your time.

MEMO: ME - 01/12/09 (1 Quotation) (Super, 09/01/12 02:53:21 PM)

P 2: 07111500 Goede.rtf:

19-19

No codes

No memos

Type: Memo

The problem with traditional approach emphasizes conformance of the system to requirements but negates the needs of the user.

P 2: 07111500 Goede.rtf - 2:2 [I'm very much against the trad..] (19:19) (Super)

Codes: [traditional approach]

Memos: [ME - 01/12/09]

I'm very much against the traditional paradigm because I believe it leads to a case where you have a system that conforms to the requirements and not to the user's needs.

MEMO: ME - 01/12/09 [1] (1 Quotation) (Super, 09/01/12 02:57:13 PM)

P 2: 07111500 Goede.rtf:

19-19

No codes

No memos

Type: Memo

This approach is suitable for developing technical applications.

P 2: 07111500 Goede.rtf - 2:3 [Object-oriented programming to..] (19:19) (Super)

Codes: [object-oriented approach,]

Memos: [ME - 01/12/09 [1]]

Object-oriented programming to me works well in the technical applications. Perhaps it's not yet proven enough for me in today's software development, except perhaps for requirement analysis.

MEMO: ME - 01/12/09 [2] (1 Quotation) (Super, 09/01/12 03:02:43 PM)

P 2: 07111500 Goede.rtf:

23-23

No codes

No memos

Type: Memo

Object-oriented approach techniques are easily understood by the users. They can therefore be used and verified with the users.

P 2: 07111500 Goede.rtf - 2:5 [use case diagrams] (23:23) (Super)

Codes: [use case diagrams]

Memos: [ME - 01/12/09 [2]]

use case diagrams

MEMO: ME - 01/12/09 [3] (1 Quotation) (Super, 09/01/12 03:06:44 PM)

P 2: 07111500 Goede.rtf:

27-27

No codes

No memos

Type: Memo

User interface can be a very good communication tool if well designed. This also can be used to improve the way people use the computer.

P 2: 07111500 Goede.rtf - 2:6 [user interface] (27:27) (Super)

Codes: [user interface]

Memos: [ME - 01/12/09 [3]]

User interface

MEMO: ME - 01/12/09 [4] (1 Quotation) (Super, 09/01/12 03:15:46 PM)

P 2: 07111500 Goede.rtf:

39-39

No codes

No memos

Type: Memo

They can be used to capture the informal part of an information system. However, there is not enough room for developers to add all their information on these diagrams.

P 2: 07111500 Goede.rtf - 2:7 [use case diagrams. You can use..] (39:39) (Super)

Codes: [use case diagrams]

Memos: [ME - 01/12/09 [4]]

Use case diagrams. You can use UML quite well. But one should not stick to the rigidness of such techniques or the prescriptiveness; one should be able to add your own things, like to be flexible. And nowadays (inaudible - noise) it's too prescribed. We don't have enough space on those diagrams to write down what its purpose originally was, and that is to capture non-technical information.

MEMO: ME - 01/12/09 [5] (1 Quotation) (Super, 09/01/12 04:02:17 PM)

P 2: 07111500 Goede.rtf:

43-45

No codes

No memos

Type: Memo

Ontologies fall into the relativistic world view and can be used to bridge the communication gap between the stakeholders in software development.

P 2: 07111500 Goede.rtf - 2:10 [Ja. Well ontologies - I'm a bi..] (43:45) (Super)

No codes

Memos: [ME - 01/12/09 [5]]

Ja. Well ontologies - I'm a bit of a philosopher as well - and it's actually an easy answer but very difficult to implement but it's between the realistic and the relativistic worldview. The realistic worldview says that everything is agreeable. You and I can decide this is a mouse and this is the function of the mouse. That is the realistic worldview and that is where the problem with the engineers comes in. They follow a realistic worldview. They think they agree on the purpose of this thing.

And if we follow a relativistic worldview, you might have an opinion on this device and I can have a different opinion on this device. And if we then get down to a discussion, we might design a much better mouse than this - if we sit down and think "what is the real purpose".

MEMO: ME - 01/12/09 [6] (1 Quotation) (Super, 09/01/12 05:06:16 PM)

P 2: 07111500 Goede.rtf:

103-103

No codes

No memos

Type: Memo

Scope creep can be reduced by getting participation or involvement of the right people, including the business sponsor. The right people involvement ensures also that the appropriate and correct requirements are gathered.

P 2: 07111500 Goede.rtf - 2:30 [I think scope creep always hap..] (103:103) (Super)

No codes

Memos: [ME - 01/12/09 [6]]

I think scope creep always happens but the extent of it depends on your - two things - actually mainly on the quality of your requirements. If your requirements are done well and you involve the right people, you might have a situation where you don't get that much scope through it. And then as - if you are using a methodology where your end-user is involved, then scope should be their problem, not your problem in a sense. But that's only the case if you have a end-user or a - I don't want to use the word end-user, I want to use a word (indiscernible - audio quality) user in data warehousing, business sponsors - somebody higher up in the organisation. Somebody more important than the secretary that's going to type on the system; somebody that has power inside the organisation. I want to have that person involved and he should prioritise all these additional scope requirements, not the programmers.

MEMO: ME - 04/23/09 (0 Quotations) (Super, 09/04/23 11:41:12 AM)

No codes

No memos

Type: Memo

MEMO: ME - 04/28/09 (0 Quotations) (Super, 09/04/28 01:06:12 PM)

No codes

No memos

Type: Memo

The documentation should be captured and stored in a repository. This will be a knowledge base upon which future problems can be referred for solutions. Think of an ontology knowledge base which also captures the human aspects of organizational information systems.

MEMO: ME - 04/28/09 [1] (0 Quotations) (Super, 09/04/28 02:09:11 PM)

No codes

No memos

Type: Memo

The different types of testing do not however ensure schedule cost issues are adhered to. They can of course ensure syntactic quality of the system.

MEMO: ME - 04/28/09 [3] (0 Quotations) (Super, 09/04/28 07:50:19 PM)

No codes

No memos

Type: Memo

MEMO: ME - 11/12/08 (1 Quotation) (Super, 08/11/12 02:21:41 PM)

P 1: 07110506 Ernest.rtf:

25-25

No codes

No memos

Type: Memo

Agile Approaches are a development methodology that uses extreme programming as a method and pair programming as a technique for XP. This PP allows a shared understanding of the development area.

P 1: 07110506 Ernest.rtf - 1:8 [For example, we are developing..] (25:25) (Super)

No codes

Memos: [ME - 11/12/08]

For example, we are developing using agile methodologies which is my area of interest. In Extreme programming you have a practice that is called Pair Programming. In Pair Programming you put two people - two programmers - on one machine and that way, that knowledge which cannot be written down but which is in those developers' minds, can actually be exchanged as they do the development.

MEMO: ME - 11/12/08 [1] (1 Quotation) (Super, 08/11/12 03:06:20 PM)

P 1: 07110506 Ernest.rtf:

33-33

No codes

No memos

Type: Memo

The software development metaphor allows people to negotiate meaning of concepts amongst themselves to achieve

a shared understanding. This is a requirement to ensure that all the stakeholders in the software development process build a language community

P 1: 07110506 Ernest.rtf - 1:14 [So instead of people being stu..] (33:33) (Super)

Codes: [Concept negotiation Technique] [Language Community]

Memos: [ME - 11/12/08 [1]]

So instead of people being stuck to those technical terms and so forth, they come up with some naming convention that is common to everyone. And that naming convention brings everyone to the same language, despite the differences and so forth.

MEMO: ME - 11/12/08 [2] (1 Quotation) (Super, 08/11/12 03:12:46 PM)

P 1: 07110506 Ernest.rtf:

31-31

No codes

No memos

Type: Memo

The software development process requires a discussion forum or a knowledge sharing platform where people can exchange their knowledge without being pestered.

P 1: 07110506 Ernest.rtf - 1:11 [whiteboards and flyers] (31:31) (Super)

Codes: [whiteboards and flyers]

Memos: [ME - 11/12/08 [2]]

Whiteboards and flyers

MEMO: ME - 11/12/08 [3] (1 Quotation) (Super, 08/11/12 03:24:54 PM)

P 1: 07110506 Ernest.rtf:

43-43

No codes

No memos

Type: Memo

To capture tacit knowledge, people require to have methods that capture the soft issues of organizational information systems. These employ the Soft Systems Methodologies that considers the humanist issues inherent in organizations. Domain ontologies can feel this gap. One can develop a domain ontology repository and use it in IS in place of the schema.

P 1: 07110506 Ernest.rtf - 1:17 [So I think the best way to act..] (43:43) (Super)

Codes: [New Development Approach]

Memos: [ME - 11/12/08 [3]]

So I think the best way to actually represent that information will be to come up with some - not a framework - but maybe language of some sort that has the ability to actually express those concepts and to relate to technical issues that way. So I'm not very sure of what language you could actually use for that but I'm tending to think in terms of the approaches to development.

MEMO: ME - 11/12/08 [4] (0 Quotations) (Super, 08/11/12 03:31:05 PM)

No codes

No memos

Type: Memo

MEMO: ME - 11/12/08 [5] (1 Quotation) (Super, 08/11/12 03:31:14 PM)

P 1: 07110506 Ernest.rtf:

47-47

No codes

No memos

Type: Memo

As much as it is an improved technique to software development, it falls short of capturing semantics. This technique which falls into the object oriented traditional paradigm however moves from functions to entities in organizational systems. It encourages reusability of objects though.

P 1: 07110506 Ernest.rtf - 1:18 [So it's more of the - when you..] (47:47) (Super)

Codes: [UML]

Memos: [ME - 11/12/08 [5]]

So it's more of the - when you look at an object, it gives a better understanding of the system or whatever you're dealing with but I wouldn't call it semantics.

MEMO: ME - 11/12/08 [6] (1 Quotation) (Super, 08/11/12 06:56:48 PM)

P 1: 07110506 Ernest.rtf:

61-61

No codes

No memos

Type: Memo

The use of a developer instead of analyst, programmer in development reduces the gap in understanding user requirements. The developer does both the analysis and programming of the system.

P 1: 07110506 Ernest.rtf - 1:25 [Ja, the approach that you are ..] (61:61) (Super)

No codes

Memos: [ME - 11/12/08 [6]]

Ja, the approach that you are defining, I generally call it a traditional approach, where you have a systems analyst, you have a programmer, and then you have got the business guy, the user. I call that the traditional approach. In the approach that I've studied in detail and also used, you'll find that the gap between the analyst and the programmer is actually closed by coming up with somebody who's called a developer, who actually elicits the requirements directly from the user. And that user, in the methods that I'm actually talking about, becomes part of the development team which means that those details - those metaphors that I was talking about - the language that is used by the team becomes familiar to that user. And the problem - the architectural problems where the systems analyst has some design diagrams and whatever that may not be clear to the programmer —

MEMO: ME - 11/12/08 [7] (1 Quotation) (Super, 08/11/12 07:09:35 PM)

P 1: 07110506 Ernest.rtf:
67-67

No codes
No memos
Type: Memo

User stories can be used for time budgeting purposes because each story can be implemented on its own. This also improves the communication between analyst, developer and users in a way established better understanding of the system.

P 1: 07110506 Ernest.rtf - 1:29 [So it is those kinds of tools ..] (67:67) (Super)

No codes
Memos: [ME - 11/12/08 [7]]

So it is those kinds of tools where the user, the analyst - whoever we call the analyst, and the programmer sits together. So sitting together there is a very important aspect or practice of eliciting the requirements or systems development. You are saying that these three guys should be together. You don't want the systems analyst to meet some guys there, get some documents signed and then go and shift that information into the programme, but you want these three levels to actually sit together. The user must be the centre of all those things. You're saying the user has to tell you the kind of requirements that you are talking about, help them mainly in terms of how to represent that information but you want them to give you that, and also to prioritise that as you develop this system, "we want you to start with the following things". And then you take those now and give them to the developer now and say "okay, let's try and find out how long it's going to take. So can you give estimates of how long each of those user cards is going to take to develop"?

MEMO: ME - 11/12/08 [8] (1 Quotation) (Super, 08/11/12 07:15:01 PM)

P 1: 07110506 Ernest.rtf:
73-73

No codes
No memos
Type: Memo

Communication medium, that is understood by both developers and users should be found and used. This really improves the understanding and the knowledge sharing in software development.

P 1: 07110506 Ernest.rtf - 1:30 [One group of clients were clie..] (73:73) (Super)

Codes: [Need for a communication medium.]
Memos: [ME - 11/12/08 [8]]

One group of clients were clients who really don't know much about systems development.

MEMO: ME - 11/12/08 [9] (1 Quotation) (Super, 08/11/12 07:30:36 PM)

P 1: 07110506 Ernest.rtf:
91-91

No codes
No memos
Type: Memo

Ontology has an open architecture and can actually be adapted to different scenarios. The ability to assign context depended meaning and hence functionality to an ontology driven case tool is a big plus.

P 1: 07110506 Ernest.rtf - 1:35 [The other thing that I would a..] (91:91) (Super)

Codes: [Requirements of a Development Tool]

Memos: [ME - 11/12/08 [9]]

The other thing that I would also mention about the competence of the tool, is that the tool should not be too much restricted to a particular methodology because, like you said, a methodology can actually change, people can come up with different approaches to that methodology, and change the framework and its parameters. So when that happens, the tool itself must actually be adaptable to that.

MEMO: ME - 11/13/08 (1 Quotation) (Super, 08/11/13 11:21:41 AM)

P 1: 07110506 Ernest.rtf:

97-97

No codes

No memos

Type: Memo

The software development approach must not be restricted to a single development tool. It will limit the acceptance of the tool itself by the users as well as negating the importance skill set variability in developers.

P 1: 07110506 Ernest.rtf - 1:36 [that you are dealing with peop..] (97:97) (Super)

Codes: [Need a Variety of Development Tools]

Memos: [ME - 11/13/08]

that you are dealing with people and people are important in those systems, and what they know is also important. It means that if you try to come up with tools like those, you are threatening the development approach itself. The methodology is threatened because you are restricting to it a specific tool within the project. Remember the projects are different and the skill sets of people that you work with are different.

MEMO: ME - 11/13/08 [1] (1 Quotation) (Super, 08/11/13 01:32:40 PM)

P 1: 07110506 Ernest.rtf:

107-107

No codes

No memos

Type: Memo

User involvement throughout the project development process in agile methodologies also ensures quality of the product. The user is able to check the conformance of the requirements and sign off the different stages on the run.

P 1: 07110506 Ernest.rtf - 1:40 [So the client must be there th..] (107:107) (Super)

Codes: [Quality Measure] [User Involvement]

Memos: [ME - 11/13/08 [1]]

So the client must be there throughout and that is basically for quality check purposes, to ensure that the kind of requirements that the user has are actually met.

MEMO: ME - 11/13/08 [3] (1 Quotation) (Super, 08/11/13 04:45:10 PM)

P 1: 07110506 Ernest.rtf:

153-153

No codes

No memos
Type: Memo

The software development fraternity should develop a tool or language that is capable of capturing the human aspect of communication. This will capture the informal part of the system into the software product. {Ontology Aspect Can help}

P 1: 07110506 Ernest.rtf - 1:55 [because the language is the on..] (153:153) (Super)

Codes: [Language Requirement] [New Development Approach]

Memos: [ME - 11/13/08 [3]]

because the language is the only solid thing or it's the only output that you get from a human being that lets you know what is happening in their brain. And if you want to deal with the informal part of the system, you are really looking at the softer issues, the human issues of software development. The behaviour of human beings and their languages would be the centre that people have to deal with in order to understand.

MEMO: ME - 11/13/08 [4] (1 Quotation) (Super, 08/11/13 04:57:01 PM)

P 1: 07110506 Ernest.rtf:

155-155

No codes

No memos

Type: Memo

This supports the idea that language limitations are the most inhibiting factor that limits software products to capture informal requirements in systems. This language should also allow the building of a language community and facilitate knowledge sharing using concepts.

P 1: 07110506 Ernest.rtf - 1:56 [For example, when somebody exp..] (155:155) (Super)

Codes: [Language Limitations]

Memos: [ME - 11/13/08 [4]]

For example, when somebody expresses their idea, the limitations of that expression or that representation, are within the language that they are using.

MEMO: ME - 11/13/08 [5] (1 Quotation) (Super, 08/11/13 04:58:45 PM)

P 1: 07110506 Ernest.rtf:

155-155

No codes

No memos

Type: Memo

Maybe since ontologies are computer processable and can capture the domain knowledge of a system, they can fill the language requirement gap discussed herein

P 1: 07110506 Ernest.rtf - 1:57 [So that is exactly the demise ..] (155:155) (Super)

Codes: [Capturing Semantics Issue] [Language Limitations]

Memos: [ME - 11/13/08 [5]]

So that is exactly the demise that we have as software developers, that if we try to go into the details of the softer side - the non-formal parts of the system - we find that we have limitations in terms of the language. Because

whatever language you are using, when I try to use now the computer language to express that, it might not represent what you are talking about. So it is really the issues of language that should be dealt with and so forth.

MEMO: ME - 11/13/08 [6] (1 Quotation) (Super, 08/11/13 05:18:53 PM)

P 1: 07110506 Ernest.rtf:

161-161

No codes

No memos

Type: Memo

Ontology concept can be used to understand meanings of concepts. It's an ontology knowledge base that can be consulted by developers in a domain and compare the different meanings from different organizations.

P 1: 07110506 Ernest.rtf - 1:58 [But what usually happens is th..] (161:161) (Super)

Codes: [Knowledge Sharing] [Language Community]

Memos: [ME - 11/13/08 [6]]

But what usually happens is that when you get to an organisation, you need to understand their language, the way they talk about things, whether they are technical or non-technical but for the purposes of your requirements. The way the information is represented will sometimes differ from other organisations even in the same field.

MEMO: ME - 11/13/08 [7] (0 Quotations) (Super, 08/11/13 05:35:23 PM)

No codes

No memos

Type: Memo

Knowledge

MEMO: ME - 11/13/08 [8] (1 Quotation) (Super, 08/11/13 05:37:24 PM)

P 1: 07110506 Ernest.rtf:

165-165

No codes

No memos

Type: Memo

If working with processes, process ontologies can be used to capture the knowledge in organizational processes and ontologies can come in place of the syntactic schema.

P 1: 07110506 Ernest.rtf - 1:61 [So what I'm saying is that you..] (165:165) (Super)

Codes: [Ontology Knowledge Base] [Process Ontology]

Memos: [ME - 11/13/08 [8]]

So what I'm saying is that you need some formal way where you can have a database that has some schemas of those words, of those concepts that we're talking about. For example, if we talk about a loan, all the financial institutions will have something called a loan. So you can have a loan schema, you find that your Nedbank, your ABSAs and Standard Banks might actually have different terms for your interest - for the interest rate, for whatever, for the borrowing rate or whatever - you know, those technical terms that they use. They might have different terms that actually mean the same thing, so your schema must be able to capture that - ensure that it is the same thing so that when you now check back and re-use it in different institutions you are not really changing much. It will come

out there according to their language but you are still using the same thing

MEMO: Need for methodology (1 Quotation) (Super, 09/01/12 08:29:20 PM)

P 3: 07111501 Magda.rtf:

160-160

No codes

No memos

Type: Memo

A methodology dictates the way a software engineering environment is subsequently used. Without that, introduces the software engineering environment on its own may cause problems of fit. This must be the case with all these ontology driven software engineering environments. The problem facing developers is a lack of a ontology driven methodology that directs most of the tools developed so far.

P 3: 07111501 Magda.rtf - 3:27 [And another thing I would also..] (160:160) (Super)

No codes

Memos: [Need for methodology]

And another thing I would also be careful though is, if companies buy into one of those software engineering environments - say for example Oracle Designer - that software engineering environment is essentially a companion to a methodology. For example, Oracle Designer, the companion methodology I would say is like information engineering. They call it something else but in essence it's information engineering. And if that methodology is not understood and in place in the company, and you use this software engineering environment, it's difficult. They find it difficult to do that. But if they know the methodology and they buy into this software engineering environment, then it becomes (inaudible - speaking softly). I call a software engineering environment and a methodology companion, those should go together. You can't have the one without the other one.

MEMO: Popular Methodologies (0 Quotations) (Super, 09/04/28 06:40:11 PM)

No codes

No memos

Type: Memo

The waterfall and V-Model are popular methodologies in the banking sector because of their repetitive and structured nature

MEMO: Problems in Software development (0 Quotations) (Super, 09/04/23 01:09:16 PM)

No codes

No memos

Type: Memo

The software development process faces a problem of user involvement due to time and business requirements of the users. More often, they do not have time to sit in the project teams and as a result, they do not understand fully what the system needs to do.

MEMO: Problems of not Involving software tester (0 Quotations) (Super, 09/04/28 08:09:51 PM)

No codes
No memos
Type: Memo

The software testers as people who check the fidelity of the implemented system to the business requirements should be involved during requirements specification. This helps them to check the feasibility of the test cases against the user requirements.

MEMO: Prominent development approaches (1 Quotation) (Super, 09/01/12 07:35:51 PM)

P 3: 07111501 Magda.rtf:
43-43

No codes
No memos
Type: Memo

Structured and object oriented approaches are widely used in industry.

P 3: 07111501 Magda.rtf - 3:9 [You mentioned only two of thos..] (43:43) (Super)

Codes: [Prominent development approaches]
Memos: [Prominent development approaches]

You mentioned only two of those approaches and in my research I saw that those two are the mostly used approaches in the industry. You find that there's a movement towards rapid application development but the structured and the object-orientated approach are indeed, I think, the most used approaches.

MEMO: Requirements problem (1 Quotation) (Super, 09/01/12 07:49:53 PM)

P 3: 07111501 Magda.rtf:
66-66

No codes
No memos
Type: Memo

Requirements are not fully captured because of communication problems as well as time limitations.

P 3: 07111501 Magda.rtf - 3:13 [And I - really I would argue t..] (66:66) (Super)

Codes: [Development Problem]
Memos: [Requirements problem]

And I - really I would argue that it's because the (client can't??) define the requirements correctly because we don't understand what exactly what this user wants. So if we put more effort into requirements gathering, I think you will see that problem - I wouldn't say it'll be solved - but much better. I think we are so much in haste to get the product delivered, that we don't even bother to look at the requirements.

MEMO: Requirements Repository (1 Quotation) (Super, 09/01/12 04:47:03 PM)

P 2: 07111500 Goede.rtf:
75-75

No codes
No memos
Type: Memo

A development tool must have a requirements repository that captures and stores user requirements during analysis, as an analysis model. These repository must be able to be consulted at every stage of the system development. Besides improving on requirements communication throughout the project, this increases also the time to market, quality of the software products.

P 2: 07111500 Goede.rtf - 2:24 [Firstly because I think it's m..] (75:75) (Super)

Codes: [Need for a development tool] [Requirements Repository]

Memos: [Requirements Repository]

Firstly because I think it's mostly - let me say, I don't think the end-user will benefit as much as the developer from doing it because you have one version of the truth of the requirement, you have one place - central repository - where you can find the requirements, you can set up checklists to see if you have fulfilled the requirements, the requirements are (nearby??) but then the system must force the developers or the analysts to fill it in.

MEMO: Requirements reusability (0 Quotations) (Super, 09/04/28 12:56:52 PM)

No codes

No memos

Type: Memo

It is difficult to reuse requirements across organizations. However, a framework of applications that run in the organizations can be development and later adapted to different organizational requirements. Such frameworks can be software kernels or SPL.

MEMO: Service oriented architecture (1 Quotation) (Super, 09/01/12 05:01:54 PM)

P 2: 07111500 Goede.rtf:

99-99

No codes

No memos

Type: Memo

This a concept or technique that encourages reuse of business models. The risk which developers are faced with is that, they tend to force business models where they can not fit.

P 2: 07111500 Goede.rtf - 2:28 [service oriented architecture] (99:99) (Super)

No codes

Memos: [Service oriented architecture]

service oriented architecture

MEMO: Software application domain (1 Quotation) (Super, 09/01/12 05:31:19 PM)

P 2: 07111500 Goede.rtf:

131-131

No codes

No memos

Type: Memo

User involvement helps to capture the context and application domain of a software product. This can be likened to a knowledge base that is populated with domain ontologies and task ontologies.

P 2: 07111500 Goede.rtf - 2:34 [Involving your users in all th..] (131:131) (Super)

Codes: [User Involvement]

Memos: [Software application domain]

Involving your users in all the stages, so that we don't spend a lot of time on things that's not important. That's I think - and I look at my students and see what they do wrong. They spend hours and hours programming stuff that the end-user are not interested in - in aesthetics and things. Aesthetics of specific functions or functionality the user is normally not interested in. Their priorities are incorrect.

MEMO: Software Crises (0 Quotations) (Super, 09/04/23 06:52:52 PM)

No codes

No memos

Type: Memo

For developers to have a product that is functional on time, they do not emphasize on the cost reduction, timely delivery, and quality requirements of system development.

MEMO: Software development methodology requirements (0 Quotations) (Super, 09/01/12 05:16:20 PM)

No codes

No memos

Type: Memo

Goede, argues that, for a development methodology to capture semantics, it must have the following characteristics: be able to capture the human factor inherent in organizational systems, direct developers to ask the right questions, must be grounded in the relativistic worldview, must incorporate tool that use the soft systems methodologies, and the techniques and tools so used must facilitate discussion and not assume understanding. Also, the methodology must allow a stratified, hierarchical way of describing a scenario. The must be several levels or ways of describing a scenario, as proposed by Dooijeweert. The various system ontologies, domain, status, process etc could be the answer to this.

MEMO: Software Migration (0 Quotations) (Super, 09/04/28 11:49:29 AM)

No codes

No memos

Type: Memo

The amount of work and tasks that are needed to migrate from one system to the other (new) discourage much organization from changing their systems. In a way, the systems are not adapted to the changing environment. This calls upon developers to look for a method that enables smooth development of adaptive software systems.

MEMO: Structured development approach (1 Quotation) (Super, 09/01/12 07:22:59 PM)

P 3: 07111501 Magda.rtf:

25-25

No codes

No memos

Type: Memo

Process Oriented and data oriented software development are referred to as paradigms. However, I would put them under the structured development approach.

P 3: 07111501 Magda.rtf - 3:3 [In the process-orientated para..] (25:25) (Super)

Codes: [traditional approach]

Memos: [Structured development approach]

In the process-orientated paradigm, the philosophical view of the developers of these methodologies is that the process is the most important aspect of the system. And the data-orientated paradigm is where they say [that] data stays constant, so we should rather focus on the data instead of the process, it would make our lives much easier.

MEMO: Studying the business environment (1 Quotation) (Super, 08/11/13 02:21:11 PM)

P 1: 07110506 Ernest.rtf:

103-103

No codes

No memos

Type: Memo

These workshops are synonymous with JAD sessions. As they are helping with knowledge sharing, they also improve the way users and developers understand the system. The important tenet is to enable analysts to understand and capture the business model of the system.

P 1: 07110506 Ernest.rtf - 1:38 [And besides that, when you do ..] (103:103) (Super)

Codes: [Developer Understanding of Business Model] [Knowledge Sharing]

Memos: [Studying the business environment]

And besides that, when you do such workshops, you are not just aiming at them understanding the system but you're also aiming at the developer's understanding of the business model that the organisation is using.

MEMO: Suitability of Waterfall approach (1 Quotation) (Super, 09/04/02 07:53:20 PM)

P 5: 07121401-Barry Myburgh Corrected.rtf:

1-1

No codes

No memos

Type: Memo

The waterfall approach is best suited for situations where there is more time for effective planning.

P 5: 07121401-Barry Myburgh Corrected.rtf - 5:1 [For situations where it is pos..] (1:1) (Super)

Codes: [waterfall approach]

Memos: [Suitability of Waterfall approach]

For situations where it is possible to clearly define what is required, and where there is enough time to do effective planning, I would follow a pretty conventional waterfall approach.

MEMO: System Developer (1 Quotation) (Super, 09/01/12 04:36:00 PM)

P 2: 07111500 Goede.rtf:
71-71

No codes
No memos
Type: Memo

Software companies are engaging the services of developers to act as communication medium. A developer has other technical and business skills and conveying messages amongst stakeholders will become easier since these people understand both languages.

P 2: 07111500 Goede.rtf - 2:23 [Now I've seen a number of orga..] (71:71) (Super)

Codes: [System Developer]
Memos: [System Developer]

Now I've seen a number of organisations employing people that has a little bit of technical skill a lot of business skills to serve as the interface between the technical people and the users.

MEMO: Time Box (0 Quotations) (Super, 09/04/23 07:15:43 PM)

No codes
No memos
Type: Memo

This is an agile development technique that fixes the time to which a software product can be developed.

MEMO: User acceptance testing (0 Quotations) (Super, 09/04/28 12:22:40 PM)

No codes
No memos
Type: Memo

User acceptance testing can be used in an attempt to make user understand what the system will do. It however comes at the end of the development stage and cannot be used to ensure that the users have an overall understanding of the whole system. In other words, it does not maintain the user understanding of the system throughout the development stages. It is in this case used as a rubber stamp of what the developers think the users wanted.

MEMO: User Involvement (0 Quotations) (Super, 09/04/23 01:03:36 PM)

No codes
No memos
Type: Memo

The client can understand the way a software product is used better if they are involved from the start of the project up to its commissioning.

MEMO: User Participation (0 Quotations) (Super, 09/04/28 07:38:30 PM)

No codes
No memos
Type: Memo

Users can be used to check the business model of the system through the system functionality

MEMO: User understanding (1 Quotation) (Super, 09/01/12 04:27:05 PM)

P 2: 07111500 Goede.rtf:

67-67

No codes

No memos

Type: Memo

These methodologies ensure that the stakeholders communicate face to face with each other. This enhances the client understanding of the would be system and eventually the quality of the software product.

P 2: 07111500 Goede.rtf - 2:22 [extreme programming and new de..] (67:67) (Super)

Codes: [agile methodologies]

Memos: [User understanding]

extreme programming and new development,

MEMO: User understanding of the system (1 Quotation) (Super, 09/01/12 04:22:41 PM)

P 2: 07111500 Goede.rtf:

63-63

No codes

No memos

Type: Memo

These techniques help ensure that the client understanding of the system is maintained throughout the development stages.

P 2: 07111500 Goede.rtf - 2:20 [Ja, I think there are two ways..] (63:63) (Super)

No codes

Memos: [User understanding of the system]

Ja, I think there are two ways and that is to do a very good verification after the requirements has been gathered. That can be done, once again, with the proof-of-concept. The other way is to incorporate the clients right through the process and not only in the requirements phase - but to talk to client's right through. Incremental development helps a lot to do delivery in smaller pieces, then if you go wrong you don't have to backtrack the whole system. So an iteration and an incremental development helps a lot. But the best way is to have a user onboard, such as all these agile methodologies, they work like this.
