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	
				Brainstorming forum	
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
				User stories	
				Proof of concept	
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.

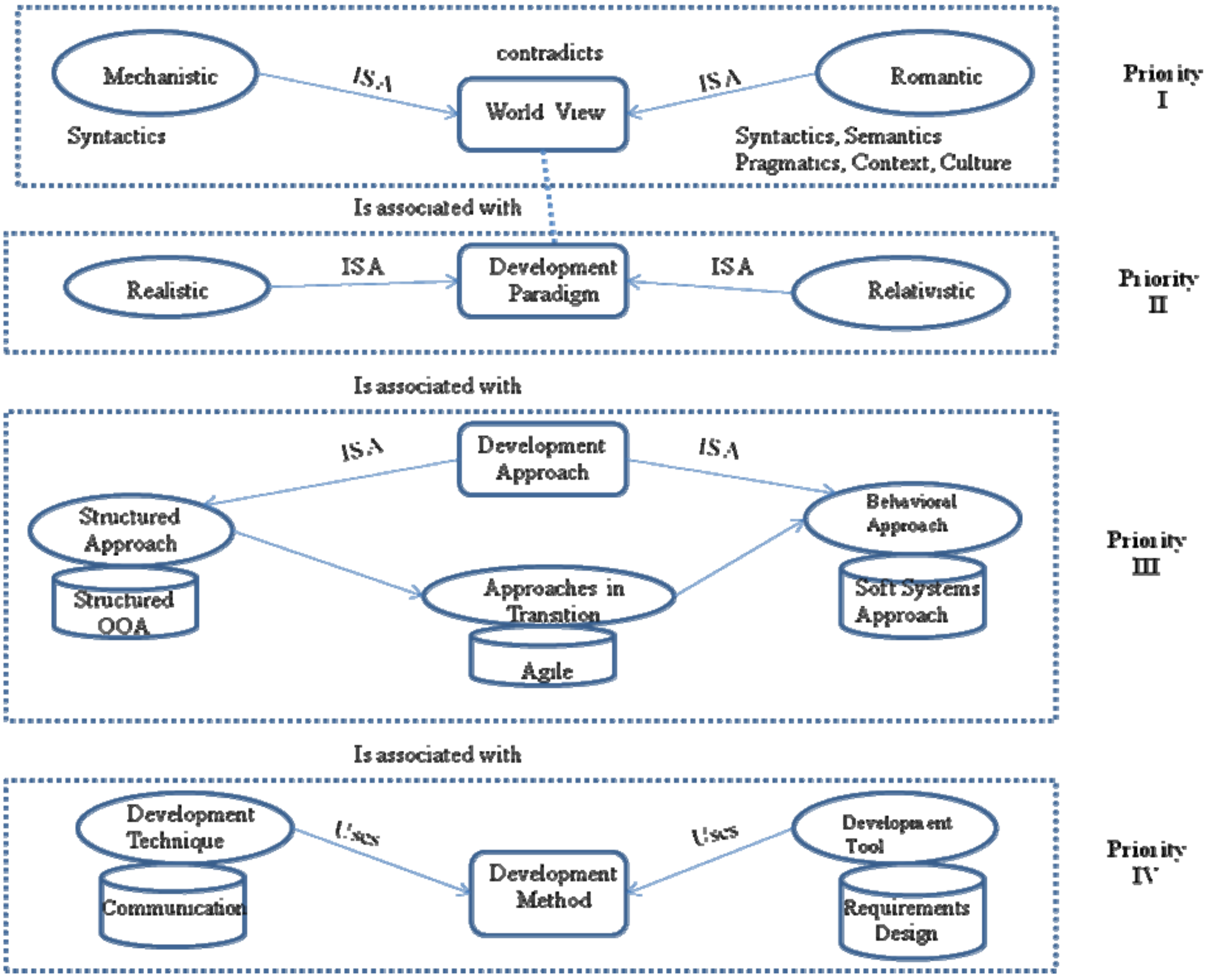

Figure 6.1: Aspects of Consideration in Software Development

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.

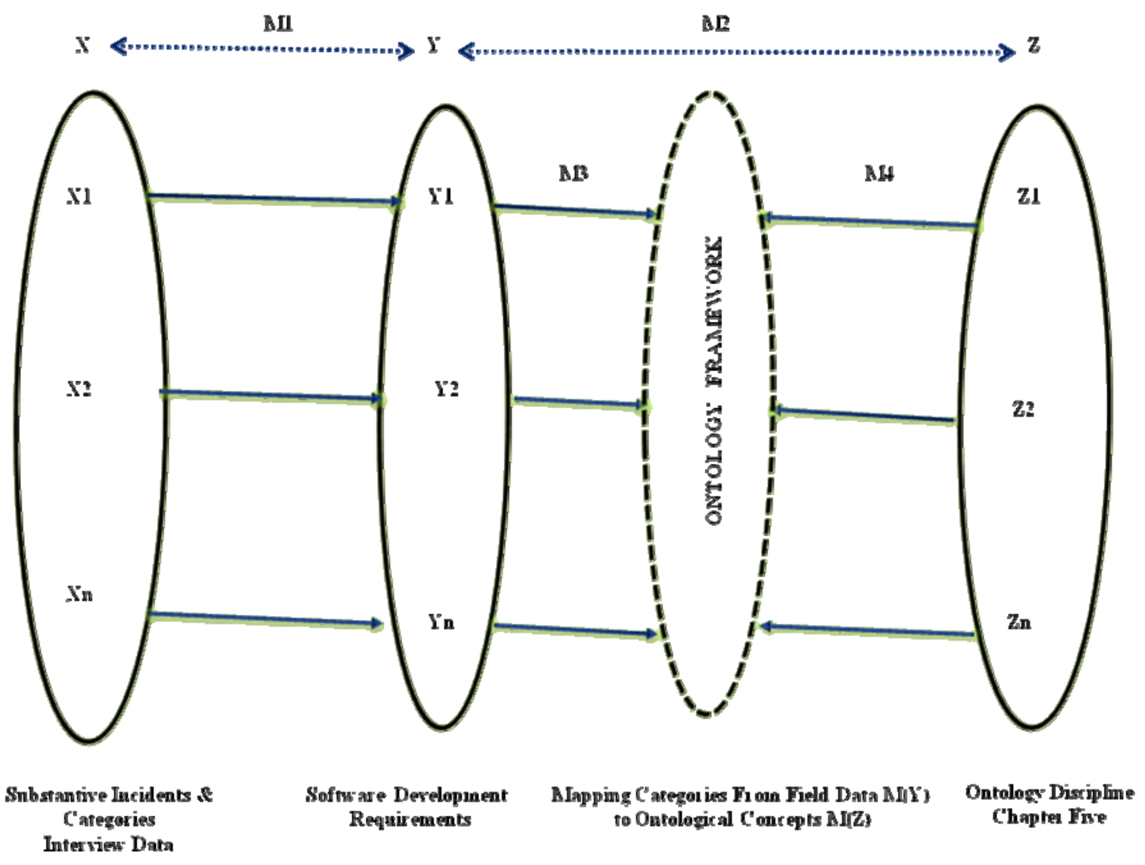
M1
M2
X
Y
Z
M3
M4
X1
X2
Xn
Y1
Y2
Yn
ONTOLOGY FRAMEWORK
Z1
Z2
Zn
Substantive Incidents & Categories Interview Data
Software Development Requirements
Mapping Categories From Field Data M(Y) to Ontological Concepts M(Z)
Ontology Discipline Chapter Five

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
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.*
Requirements: • 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. These are supported with evidence from section 6.6.1

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.*
Requirements: • 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 (*supported by Section 4.1.1).* This is further supported with evidence from section 6.6.2.
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 tool that captures the organizational context during analysis and maintains it through the subsequent development stages and the development life cycle of the system.*
Requirements: • 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 *(supported by Section 4.1.3).*

- 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 Oliviera (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 *(Section 5.6.1).*
	Maintain business model and domain model characteristics from analysis to implementation.	Use of domain ontologies as software model *(Section 5.6.1).*
2	Capture possible life states of a system.	Ontology is an intensional model of the system (*Section 5.4.1).*
3	Capturing system requirements-requirements specify the processes that run in organizational systems (specification model).	Use process/method/task/activity ontologies *(Sections 5.6.2 and 5.6.6).*
	Capture specification model.	Method and process ontologies capture the knowledge and reasoning needed in performing a task *(Section 5.6.2).* 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 *(Sections 5.6.2 and 5.6.6).*
5	Avoid errors during software development. Improve software quality.	Domain ontologies carry the domain knowledge into the software product itself *(Section 5.2)* 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 *(Section 5.6.2).*	Use domain, process or method ontologies *(Sections 5.6.1, 5.6.2 and 5.6.6).*
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 *(Section 5.6.3).*
8	Capture behavioural aspects of the system.	Use intentional ontologies –they model ascriptions of intentions to actors in a system *(Sections 4.6.4 and 5.6.4.)*

9	Capture organizational culture and context.	Use social ontologies – these model organizational structure and their interdependencies (*Section* 5.6.5*),* such as roles and responsibilities.
10	Reuse of ontology-driven analysis model.	Use ontology-driven repository *(Section* 5.8*).*
11	Reuse of software requirements.	Use ontology-driven repository *(Section* 5.7.5*).*
12	Ensure quality, reduce cost and keep to scheduled times.	Use ontology-driven software development environment *(Section* 5.8*).*
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.		

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 (Uschold 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	• 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	• 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	• 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

Process that drive Interpretations.	sufficient time and resources to study the organizational culture, context and environment. • 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.