An Ontology-Driven Software Development Framework

Nehemiah Mavetera, North-West University, Mmabatho, South Africa,
Nehemiah.mavetera@nwu.ac.za
Jan Kroeze, North-West University, Vanderbijlpark, South Africa, Kroeze.Jan@nwu.ac.za

Abstract

The software development process has been curtailed by the lack of a methodology that can capture and maintain the softer, humanist characteristics of organizational systems into the software product. This is attributed to the absence of a software model that can capture and maintain these characteristics at analysis through to design and implementation phases of the development life cycle. Using grounded theory method, the authors investigated issues that limit the usability of software systems in organizations. These were tracked back to the developmental stages of software products and were attributed to the human aspects of organizational systems that are not captured. On the other end, ontologies are explored and positioned as artefacts that can be used to capture the softer, human aspects of organizational systems.

This paper therefore presents a framework that positions ontologies at the centre of the software development process. This ontology artefact takes the role of the software model that bridges the communication gap between the software development phases as well as among stakeholders in the development process. At the same time, it allows soft issues such as culture, social context, semantics and pragmatics to be maintained in the software products that run organizational information systems.

Keywords: Ontology, Software development, Methodology, Grounded theory method

.

Introduction

Organizations operate in environments that have their own culture, social context and a defined practice. It is therefore paramount to keep the human aspect of these organizations in the information systems and software products that are used to represent them. The biggest problem in maintaining these human aspects is noted during the communication process during the development stages of information systems. Most importantly, during analysis and design phases, the methods used in capturing the analysis and design models overlook the importance of capturing and maintaining these human behavioural characteristics of organizations.

This is a problem in software development, where a socially constructed; semantically enabled organizational system is forcibly mapped onto a syntactic software product. In addition, there does not exist a "*common vocabulary or world view*" that can be understood and used by all participants in the development process. Gonzalez-Perez and Henderson-Sellers (2006) hence depict the software development process as a complex activity with a varied number of stakeholders.

Furthermore, most of the problems encountered in software development fall within the deficiencies of the methodological approaches, that is the approaches, methods, techniques and tools that are used during the development of software products. With the advent of ontologies and their use in software development, many researchers have quickly moved to incorporate these artefacts in the production and use of software products.

From the ontology research side, several works have been done on ontology development, representation languages such OWL, DAML, and protégé. Tools for ontology building and mapping, semantic mark-up languages and mark-up of resources such as web services have also been developed. On the interoperability side, ontology-enabled, semantic interoperability frameworks (Mavetera 2004; 2007; Lemmens 2006) have also been suggested and some implemented.

Despite all these developments, there is very little evidence of research on the methodological implications of introducing ontologies in software development. The biggest question is: What is the role of ontologies in software development? How can ontologies be incorporated in software development? What characteristics of ontologies are important for the development of software products and their subsequent use in information systems? As a result of advocating for their use in software development, researchers are forced to ask the methodological implications of incorporating ontologies in software development. All these are questions begging for answers from researchers and software practitioners alike.

As a starting point, this paper discusses the problems that are found in the field of software development, their causes and suggests a software development framework that can be adopted in order to reduce their impact on the software development process. In coming up with the framework, the paper also explores the characteristics of ontologies and the role they can play in software development. At the end, a framework that can be used to improve the software development process is presented. Throughout the paper, unlike classical ontology from metaphysics, ontology will be regarded as *"a linguistic model of the world that comprises syntactics, semantics, pragmatics, as well as the social context of that which is represented. Despite some unavoidable informal indeterminacy in the real world view, it should allow a shared, descriptive, both structural and behavioural modelling and representation of real world view by a set of concepts, their relationships and constraints under the open-world assumption."* In this case, the open world assumption principle holds that anything that is not explicitly expressed by ontology is therefore not known.

The rest of the paper is as follows: Firstly, a brief of software development issues will be made and ways of improving the software development process are proposed. Secondly, a discussion of some possible uses of ontologies in software development is made before giving an outline of the research methodology used in this study. Next, the proposed ontology-driven framework for developing software products is presented while the concluding part looks at the implications of this framework in the software development field.

Software product development issues

Software development is an attempt to map organizations to software –driven systems. Most of the software development methods in use are targeted at improving the production rate, quality and maintainability issues of the software products. Some of these practices are software kernels and software product lines (Dittrich & Sestoft 2005).

However, the product line approach is a predictive type of software reuse strategy, where software artefacts can only be packaged, if there exists a chance that they may be used in one or more products that are in the same product line(Krueger n.d.). The failure of the product line approach is not attributed to the production rate but the lack of adaptivity of the kernels developed and so used.

Communication in software development

Another issue that warrants discussion is that of communication. Communication is very important in software development especially at analysis stage. This stage is supposed to produce an analysis model that is descriptive and is in the form of a computationally independent model (CIM) (Aβmann *et al.* 2006). The computationally independent viewpoint captures the system environment and the system requirements. There is however a problem of maintaining system characteristics and later transferring them to the design and implementation models. This linguistic communication problem is depicted on Figure 1 below.

It must be noted that the analysis model is derived from the organizational system's environment and is intended to capture the triplet, domain model, business model and the requirements model. The requirements model manifests itself as the system specification. The design model is the architectural model of the system, and at this stage, it must capture the system from the designers' viewpoint but still must be platform independent. The implementation model is then gradually populated with platform specific details as discussed in Aβmann *et al.* (2006).

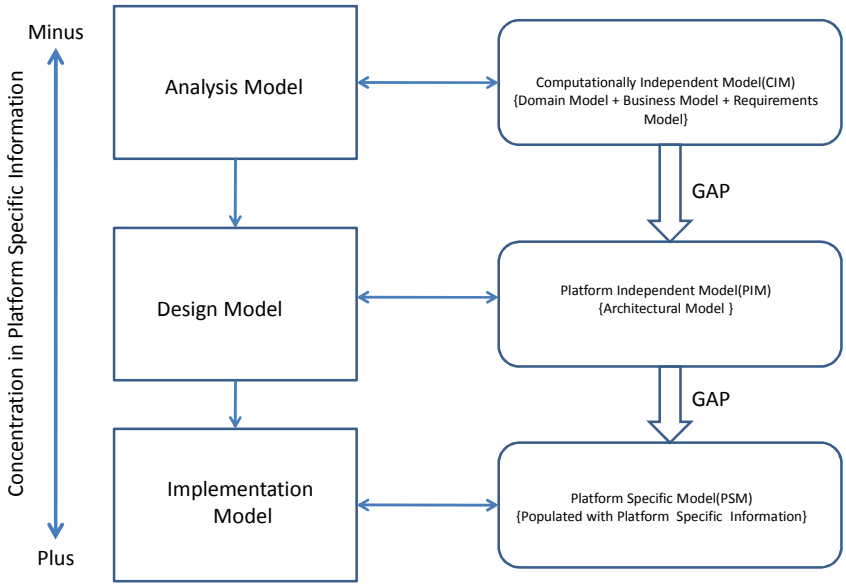

Figure1. Communication Gap in Software Development

However, due to developers' insistency on formalization, from the analysis model through to the implementation model, platform specific information is always allowed to creep into the system. Also, there is a problem of translating all the characteristics of the analysis model (CIM) to the implementation model through the design model (PIM). One reason is that, the system requirements are translated to a specification model (SM) at the end of the analysis stage. This SM is only an instantiation of parts of the functions of the system.

In order to address this problem, (Gunter *et al.* 2000) developed a reference model that could ameliorate the problems of communication between the environment, system, and the medium. The model could map the environment to the software system through a software model thereby maintaining the fidelity of system characteristics in the environment to the developed system. This in theory has been proposed but no methods, techniques or tools have been found that can be used as this software model.

Ways of improving the software development process

It is accepted that each development process requires guiding principles, methods, tasks and activities that are guided by a specific body of knowledge. A software development process, life cycle or software process can be conceived as a *"structure imposed on the development of a software product"*. Several issues have been identified that can be of relevance in improving the software development process. Most of them have to address the issue of communication while at the same

time looking at the overall software development practice. In order to improve the software development process, in addition to Roque *et al's* (2003) requirements, the software development process must:

- Allow software developers to frame their activities, supported by the relation between context and mediators of the activities that mould that context.
- Achieve an understanding of the software development (SD) activities on the proposed framework, viewing information systems (IS) development as a social-technical phenomenon within a cultural and historical envelope.
- Capture semantics, pragmatics and context in the software product.

This discussion enforces some requirements to any software development model that include but are not limited to the following:

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

Since the model is tasked with the transcription of possible system states to the software product, the model must therefore have sufficient obligatory passage and translation points (Introna 1997; Mavetera 2004). The obligatory passage and translation points allow the mediation of different contexts for different organizational software products.

Reflecting through the discussion of this paper, we discover that software development approaches, from the beginning have been grounded on philosophical frameworks that use the realist world view as their cornerstone. This worldview, in conjunction with the functionalist paradigm, views the served system separate from the serving system this grounding has allowed the development of mechanistic software products. Mechanistic software products rely and are based on the ability of computer systems to represent and arrange the signs that are used in a language. As Tarnas (1991) added, in these software products, "the material particles captured and represented" possess "neither purpose nor intelligence". This paper motivates for a shift from the current mechanistic software development approaches to the one that allows the development of romantic software products. Romantic products are free from the dictates of syntactic machine representations. The romantic world view considers the "world as a unitary organism" and this is contrasted to the rational atomistic view of the mechanistic world (Tarnas, 1991).This approach is grounded in the relativistic paradigm. To move to this relativistic paradigm, the following issues have to be considered:

- There must be a switch from hard systems paradigm to soft system approaches.
- The methodological approach must have a way of capturing the dynamic nature of the ever running context in organisations. On this note, the approach should ensure that software developers are able to study organizational environment, live in it so that they can have a

situated practice and experience this practice before embarking on any software development project.

- The approach should ensure a transition from task based approach to role based approach.
- Developers must be able to build a language community with all stakeholders, that is, there should be a linguistic model that is used to negotiate a shared understanding of the concepts found in a system. This requirement supports the need for improved communication methods, techniques and tools that can be used during the development process.
- The software product must be able to capture semantic and pragmatic (tacit) information in organisations.
- In addition to being able to model behaviour, the organisational culture and context have to be captured. These aspects are always an integral part of organizational behaviour.
- In the current modern and pervasive computing environments where software development is outsourced offshore, the development approach should have a platform for different developers to share their understanding of the system requirements regardless of their locations.
- The functional requirements and the system requirements must be mapped from the organizational environment to the systems platform through a software model that does not negate the social or human aspects of the organisational system.
- As de Oliveira *et al.* (2006) explain, a common repository, a guiding framework to the software process, domain and task knowledge are prerequisites for a sound software development environment. These should be captured in a software development environment (SDE). The repository is a store for all information related to the software development life cycle (SDLC). In addition, each software development process requires knowledge about the organization. This knowledge sets the context of the software product.
- Data and experience gained from previous software projects, in addition to identifying relevant key personnel, that can work on a project should also be acquired.
- An analysis model, derived from the domain theory must be designed for a family of systems in the same domain. This analysis model should be translated to the design and implementation models without any loss of its attributes and descriptiveness.

This discussion, together with the software development requirements discussed in Mavetera and Kroeze (2009b), has highlighted the problems with software development process and also proposed what is needed to have a shift in the current development methods. In the following section, a brief discussion on ontologies and their characteristics will be made.

Possible uses of ontologies in software development

Ontologies can be regarded as domain related intensional models. There are however different ways of classifying these ontologies. One can classify them according to their granularity (Guarino 1998), functions or form (Fensel 2004), type of conceptualization structure (Van Heijst et al. 1997) or the nature of existence which Jurisica *et al.* (1999) termed the "*nature of the real world issue*" that needs modeling. Ruiz & Hilera (2006) in their discussion noted the existence of another classification criterion that considers or combines two or more of the abovementioned categories. The bi-dimensional classification proposal as they called it depends on the "level of richness" of the ontology's internal structure and the "*subject of the conceptualization*" (Ruiz & Hilera 2006). In other

words, it is based on the architectural perspective of the ontology. This section will look at the different ontology types that can be used in software development. Some of these ontology types are domain ontologies, method ontologies, status ontologies, that combine static and dynamic ontologies as proposed by Jurisica *et al.* (1999), intentional ontologies, social ontologies and finally process ontologies.

Domain ontologies

In any study discipline or domain, there exists generic or specific knowledge to be captured. This depends usually on the requirements of the developers. Domain ontologies as they are known capture knowledge valid for a particular type of domain. They have to be reusable and a knowledge base for concepts related to a domain and their relationships should be developed.

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

The analysis model is portrayed as comprising of the domain, business model and requirements model. This analysis model is a descriptive type of a model that conforms to the open world assumption. The domain ontologies can therefore be used to model the domain and business models during software development.

Method ontologies

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

Status ontologies

In software development, we recognize the need to capture and represent the static or the dynamic characteristics of a system. Therefore there must exist both static and dynamic ontologies. These ontologies represent the status of an artefact. Status ontologies as they are called herein argue for a world with artefacts that exist and do not change their form of existence (static) and also another class of things that change with time (dynamic). As such dynamic ontologies are used to abstract the behavioural characteristics of a system. Their characteristics consist of "*concepts and the interrelationships"* (Aβmann et al. 2006). Each software development process should allow the modeling of both static and dynamic states of a system.

Intentional ontologies

These ontologies are intended to model the softer aspects of living things. These are the sort of things that can have beliefs, desires and intentions (BDI). In this category, the human aspects of living things are modelled and examples of such ontologies are aspect, object, agent and support as stated in Ruiz & Hilera (2006). These types of ontologies are also meant to model ascriptions of intensions to actors in a system.

Social ontologies

Social ontologies describe the organizational structure and the interdependencies that exist among the social actors in these organizations. At a high abstract level, they include concepts such as actor role and responsibility to mention but a few. In software development, actor roles such as analyst, developer, tester to mention but a few can be used as examples.

Process ontologies

In most system development projects, requirements specify the processes that run in those systems. As Haller & Oren (2006) noted, the process modeling activity has been used to describe the dynamic behaviour of organizations as well as capturing the processes and the context in the organization. The organizational context includes data, roles played in the organization and the resources that are utilized in the process. These issues, captured as requirements are usually represented using the specification model. The specification model is very prescriptive in nature and hence conforms to the closed world assumption.

Process ontologies can capture the three aspects of enterprise knowledge, that is, context, content and the structure. These are reflected as the enterprise knowledge, the domain knowledge and the information knowledge respectively. The enterprise knowledge consists of processes, organizational structure, IT structure, products and customers. The domain knowledge comprises of the terms, concepts and their relationships and lastly, the information knowledge consists of document types and document structures that are used in the organization. Process ontologies can capture this triplet as a meta knowledge model for the organization (Haller & Oren 2006). Hence, the addition of process ontologies as a specification model may remove the closed world assumption nature of the specification model.

Process ontologies and method ontologies can be regarded as fulfilling the same role in software development. Also of note is the fact that a process repository can be used to capture process models, business rules, and checklists, application data and document knowledge, and the background knowledge.

Research method and design used in this study

In this study, Grounded Theory Method (GTM) was used as the research method. GTM is a generative method whose purpose is to generate or produce theory (Trochim n.d.). This is supported by Hacking (2002) who argued that for theory to exist and later be deductively tested, it must first be developed. In this study, the authors intended to gain an understanding in the field of software development and organizations. In the investigation, they used open interviews to gather data from seven (7) practitioners in the field of software development. The first three (3) interviews were used for open coding in order to generate hypothesis and research questions. Later, four other interview data samples were gathered from a project manager, systems analyst, software developer and lastly, a software test analyst.

This second group of data samples were used in the study to improve the quality of the generated theory and to reach its theoretical saturation (Glaser & Strauss 1967; Strauss & Corbin 1990). The literature on software development and ontologies discussed above was also used in order to map the software development concepts to their ontology equivalents as shown in Figure 2.

The double mapping principle

Figure 2 shows the double mapping principle used in this study (Mavetera & Kroeze 2008). This mapping was necessitated by the presence of two substantive areas in this investigation. In brief, the

researchers needed to map occurrences in the field data to some concepts in the first substantive area, the software development study area, that is, mapping M1 in Figure 2. This first mapping, M1, does not fully address the requirements of the research that is to come up with "An ontology-driven software development framework". This therefore necessitated the need for a second mapping M2.

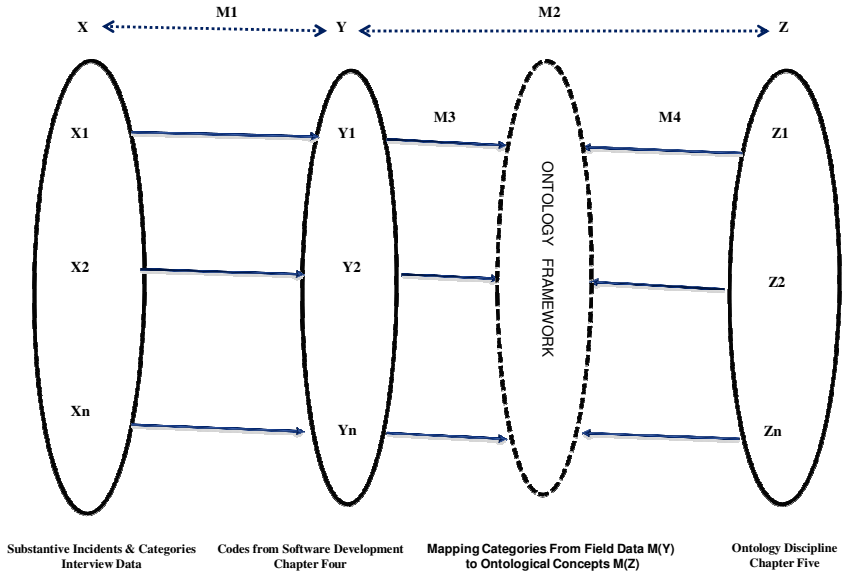

Figure 2. Expanded Double Mapping Principle (Adapted from Mavetera & Kroeze, 2008)

Using Figure 2, the first mapping M1= M(X) to M(Y) maps the incidents and categories of incidents from the field data M(X) to concepts in the software development field M(Y). This M(Y) is a complete list of software development requirements elicited from the data analysis of all the seven interview data samples as well as literature study on software development. This is reflected in Table 1 as the "software development requirements" column.

The second mapping, M2=M3+M4 is required to map the software development requirements categories M(Y) to some set of ontology concepts that they relate to, M (Z), that is, the transformations M3 and M4 in Figure 2. As already discussed, 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. These ontology elements do not directly have the same meanings as the concepts in use in current software development practices. More importantly, the interviewees did not understand the field of ontologies as discussed in Mavetera and Kroeze (2009a).

In this mapping, it must be emphasized that the mapping M3 is a reflection of M(Y) and M4 is also a reflection of M (Z), that is, the ontology characteristics as derived from the study of ontology literature. This ontology literature was used as a secondary source of data. Mapping M2 therefore matches a set of ontology elements M (Z) through M4 to a set of software development requirements M(Y) through M3. The result of this mapping is reflected in Figure 2 as "the ontology framework" for use in software development. Table 1 below shows the results of mapping M2 and these constitute the ontology-driven software development framework.

Ontology driven software development framework

Table 1, shows three columns, that is, the issue under consideration, the software development requirements that need to be addressed, and lastly, the ontology characteristics that can be incorporated in order to address this requirement. It must be noted that, for each requirement, developers are urged to develop ontologies that satisfy the said requirement and store them in a software development environment. These ontologies can then be used as the basis upon which the human aspects of organizational information systems can be captured, stored and reused during software development. For example, in order to capture and maintain the business and domain model characteristics of organizations from analysis to implementation, domain ontologies can be developed and used during software development. This applies to each and every requirement listed in table 1.

Table1: Ontology Driven Software Development Framework

Issues	Software development requirements	Ontology aspects
1	Maintain business model & domain model characteristics from analysis to implementation	Domain ontology as software model
2	Capture possible life states of a system	Ontology is an intensional model of the system
3	Capturing system requirements- requirements specify the processes that run in organizational systems (specification model)	Process/method/task/activity ontologies
4	Capture the requirement specification model in a domain related language	Method or process ontologies
5	Capture specification model	Method & process ontologies capture the knowledge and reasoning needed in performing a task. 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.
6	Avoid errors during software development	Domain ontologies carry the domain knowledge into the software product itself through to maintenance stage.
7	Capture domain & business model	Domain ontology
8	Capture the descriptive analysis model- the analysis model is a descriptive type of a model that conforms to the open world assumption	Domain, process or method ontologies.
9	Capture behavioural attributes of systems, i.e. static and dynamic attributes. Software development must allow for modeling of	Status ontologies can capture the static (change in form of existence) and dynamic (time dependent) aspects of

	both the static and dynamic states of a system.	organization.
10	Capture system behavioural aspects	Intensional ontologies –they model ascriptions of intensions to actors in a system
11	Capture organizational culture and context	Social ontologies- these model organizational structure and their interdependencies such as roles and responsibilities.
12	Reuse of analysis model	Ontology driven repository
13	Reuse of software requirements	Ontology driven repository
14	Ensure quality, reduce cost and keep to scheduled times.	Ontology driven software development environment
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 store and maintained in an ontology driven repository. For addressing the requirements of the software development metrics that is improving schedule times, quality and reducing development costs, an ontology driven software development environment must be used.		

There are however several practical applications that ontologies can be used during software development. This framework is therefore not exhaustive. As Sarantakos (1997) noted, a framework like this is supposed to emerge from experience and is continuously revised and corrected through several research studies. Most importantly, it must not act as a "blinder or straight–jacket". It must be directed and fine tuned to serve the needs of software developers.

Developers must note that there exist an abundance of application areas for the ontology driven software development framework like this one. The framework as presented here lacks some application context in terms of a formalized way of using it. It therefore requires an ontology driven software development approach coupled with a methodology for it to be used in every day software development process.

Conclusions and future work

This paper discussed problems that are encountered when developing organizational software products. Featuring the most is the problem of communication in organizations and during the software development process. The paper also discussed the field of ontologies and portrayed ontologies as linguistic models that can be used to capture organizational culture and social context. At the same time, ontologies can be used to address and solve most of the problems encountered in organizational software and system development processes. At the end, a framework of ontology components was presented.

This study is a reflection of aspects that often pose problems in organizational systems. The human aspect of organizations must always be maintained in the software products that are developed and used to represent the organizational information systems. As future work, the study will use the framework presented here as a starting point to the development of an ontology-driven software

development methodology. And most importantly, the various ontology types proposed here have to be designed and developed. These will be grouped into a software development environment for a specific study discipline. As the reader can note, this is not the end, but just the start of developing a humanist enabled, ontology-driven software development approach. The methodology will be a moderate attempt at developing some formal theory from the substantive theory discussed in this paper.

Acknowledgement

This material is based upon work supported financially by the National Research Foundation.

References

Aβmann, U., Zschaler, S. & Wagner, G. (2006). Ontologies. Meta-models and the Model Driven paradigm. Ontologies for Software Engineering and Software Technology. Coral Calero, Fransisco Ruiz and Mario Piattini (eds), Springer Verlag,

De Oliveira, K.M., Villela, K., Rocha, A.R. & Travassos, G.H. (2006). Use of Ontologies in Software Development Environments. Ontologies for Software Engineering and Software Technology. Coral Calero, Fransisco Ruiz and Mario Piattini Eds) (eds) Springer Verlag.

Dittrich, Y & Sestoft, P. (2005). "Designing Evolvable Software products: Coordinating the evolution of different layers in Kernel Based Software Products". [Online]. [Retrieved October 2006]. http://www.itu.dk/research/sdg/doku.php.

Fensel, D. (2004). Ontologies: A Silver Bullet for Knowledge Management and Electronic Commerce, Second Edition, Springer Verlag, Berlin, Heidelberg.

Glaser, B, G. & Strauss, A, L. (1967). The Discovery of Grounded Theory: strategies for qualitative research, Aldine De Gruyter, New York.

Gonzalez-Perez, C. & Henderson-Sellers, B. (2006). An ontology for software development endeavors. Ontologies for Software Engineering and Software Technology. Coral Calero, Fransisco Ruiz and Mario Piattini (eds), Springer Verlag.

Guarino, N.(1998). "Formal Ontology and Information Systems". *Proceedings of FOIS'98,* Trento, Italy, 6-8 June, 1998. Amsterdam, IOS Press, p.3-15.

Gunter, C. A., Gunter, E. L., Hill, M. N. J. & Zave, P. (2000). Formal Software Engineering. *ACM. SIGSOFT. Software Engineering Notes*, Vol. 25(1), pp. 54, January 2000.

Hacking, I. (2002). Historical Ontology. Harvard University Press, London.

Haller, A. & Oren, E. (2006). A Process Ontology to Represent Semantics Of Different Process and Choreography Meta Models, DERI Technical Report, National University of Ireland, Galway.

Introna, L.D. (1997). Management Information and Power, Macmillan Press.

Jurisica, I., Mylopoulos, J. & Yu, E. (1999). "Using Ontologies For Knowledge Management: An Information Systems Perspective". *Proceedings of the 62nd Annual Meeting of the American Society for Information Science (ASIS99),* 482-496.

Krueger, C. W. (n.d.). "Introduction to software product lines". [Online]. [Retrieved 17 June 2008] http://www.softwareproductlines.com/introduction/introduction.html..

Lemmens, R. (2006). Semantic Interoperability . PhD Dissertation, ITC, Netherlands. [Online]. [Retrieved 17 June 2008] http://www.ncg.knaw.nl/publicaties/geodesy/pdf/63lemmens.pdf.

Mavetera, N. & Kroeze, J. (2008). "Practical issues in Grounded Theory Method Research', *Proceedings of the M & D Graduate Symposium, preceding SAICSIT 2008,* 6 October 2008, Wilderness, edited by C. Cilliers, L. Barnard and R. Botha. Port Elizabeth: NMMU.

Mavetera, N. & Kroeze, J. (2009a). Practical considerations in Grounded Theory Method Research. *Sprouts: Working Papers on Information Systems*, Vol. 9 No. 32. [Online]. Available: http://sprouts.aisnet.org/9-32, ISSN 1535-6078

Mavetera, N. & Kroeze, J. (2009b) "A Grounding Framework for Developing Adaptive Software products" *Proceedings of the 13th IBIMA conference on Knowledge Management and Innovation in Advancing Economies*, Marrakech, Morocco, 9-10 November 2009. ISBN 978-0-9821489-2-1.

Mavetera, N. (2004). "The Philosophy of Ontologies: A new Information Systems Development Paradigm" *Proceedings of the International Science and Technology Conference (ISTC'04),* 1-2 December 2004 , Vanderbijlpark, SA, ISBN: 0-620-33202-6.

Mavetera, N. (2007). "A Comprehensive Ontology-Driven Software Development Architecture: A Holistic Approach to Developing Romantic Software Products". Managing Worldwide Operations and Communications with Information Technology. *Proceedings of 2007 Information Resources Management Association, International Conference,* Vancouver, British Columbia, Canada, May 19-23, 2007), Khosrow-Pour (eds). IGI Publishing, Hershey, PA. (CD ISBN: 978-159904930-4.) (Hardcover ISBN: 978-1-59904-929-8.)

Roque, L., Almeida, A. & Figueiredo, A. D. (2003). Context Engineering: An IS Development Approach, *In Proc. of the Action in Language, Organisations and Information Systems,* ALOIS'2003, Linköping, Sweden, 2003, pp. 107-122,

Ruiz, F. & Hilera, J.R. (2006). Using ontologies in Software Engineering and Technology. In Ontologies for Software Engineering and Software Technology. Coral Calero, Fransisco Ruiz and Mario Piattini,(Ed), Springer Verlag,

Sarantakos, S. (1997) Social Research, 2nd ed,Palgrave, New York.

Strauss, A. L. & Corbin, J.(1990). Basics of Qualitative Research: Grounded Theory Procedures and Techniques, Sage , London.

Tarnas, R (1991) The Passion of the Western Mind. Understanding the ideas that have shaped our world view, Pimlico, UK.

Van Heijst, G., Schereiber, A.T. & Wielinga, B.J. (1997). "Using Explicit Ontologies in KBS Development". *International Journal of Human and Computer Studies*, Vol. 46, No. 2(3) 293-310.