IBIMA Publishing
Communications of the IBIMA
http://www.ibimapublishing.com/journals/CIBIMA/cibima.html
Vol. 2010 (2010), Article ID 340296, 15 pages
DOI: 10.5171/2010.340296

Guiding Principles for Developing Adaptive Software Products

Nehemiah Mavetera and Jan Kroeze

North-West University, South Africa

Abstract

This paper presents an analysis of problems that are faced by software development practitioners. While communication, team management, coding and software documentation are some of the persistent problems, the first and major troubles for software practitioners is to select appropriate software development approach. This approach should allow developers to develop adaptive software products. The role of organizational culture, context, practice and concepts in developing adaptive software products is also discussed. This qualitative research study interviewed seven software development practitioners in South Africa, focusing on software developmental methods that allow the capturing of softer, human elements inherent in organizations and the accompanying problems that inhibit their inclusion in the resultant software products.

The research used Grounded Theory Method, to construct a framework of requirements that must be considered when choosing a software development approach that allows the development of adaptive software products. This framework highlights the importance of employing a software development approach that is grounded in the relativistic paradigm, adopting a behavioral systems approach and adopting methods whose communication techniques and tools can capture the humanist elements that are inherent in organizational systems.

Keywords: Software Development, Framework, Adaptive Systems, Approach

Introduction

A software development process is required to come up with a piece of software. This software development process can be viewed as a framework or structure that is used during the development of a software product. The field of software development is characterized by two major groups of stakeholders: the software developers and the method engineers (Gonzalez-Perez and Henderson-Sellers, 2006). Software developers may be in any one of the following categories: system analysts, programmers, business analysts or system architects. On the other hand, method engineers define and prescribe a methodology that is used by developers in their quest to construct software products. These two terms, that is, software developers and method engineers, refer to the roles played by either individuals or organizations involved in the software development process.

However. the software development process is faced with a myriad of requirements that, over the years, have been found to be very difficult to satisfy. Examples of such requirements are the

 Contact author: Nehemiah Mavetera. E-mail: Nehemiah.mavetera@nwu.ac.za

ability to capture organizational culture, context, practice, the concepts used in an organization and the ability to develop adaptive software products. More overarching is the failure to capture the human element inherent in organizational systems into the software product. In addition, in the development of software products, organizational knowledge is a key factor. This knowledge is found in the organizational structures of the system to be developed. Since information systems must capture and share this knowledge, the tacit nature of the knowledge makes it difficult for system developers to totally understand the task of software product development.

The rest of the paper is as follows: The research method used in this study will be discussed, Followed by a discussion on software development issues as reflected in the literature on software development. Effective communication is highlighted as a major driver for successful software development. As such, communication issues that are faced by software developers and which method engineers should address when formulating software development approaches are addressed. Using data from the empirical study, practitioners' viewpoints on issues that should be considered during the development of software products will be highlighted. These viewpoints lead to some propositions that need to be addressed using the guiding principles for developing adaptive software products as proposed herein. The research study employed Grounded Theory Method (GTM) and the reporting of the study will follow the sequence of processes that were undertaken using this method.

Research method and design

"adequacy of a theory ... can not be divorced from the process by which it is generated".

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

Every research process, whether deductive or inductive type has to use some theory. As Glaser and Strauss (1967) put it, if theory generation is the goal of research, the processes undertaken in such an endeavor should be chosen, planned, executed carefully and documented comprehensively. These processes will contribute immensely to the acceptance and final use of the theory in practice.

When researching phenomena that do not fall into the functionalist and positivistic paradigms, researchers find it compelling to spend some time in choosing the most appropriate research methods. Most importantly, if the research is qualitative and interpretive in nature, greater emphasis should be placed on the research design process. This is an attempt to ensure that two factors, that is, quality and rigor, are satisfied.

More often, the philosophical nature of a research problem dictates the research approach and methodology that will be followed. To borrow Glaser (1992)'s dictum, a methodology can be described as "*a theory of methods*". A single research project may use one or several methods. For any particular research project, it is therefore of paramount importance to order and describe the research methods in a coherent way.

In an interpretive research like this one, the researcher required to constantly modify the data-gathering process as the study progresses (Trochim, n.d.). This is in response to the changing environment and the understanding of the researcher as the study progresses. In fact, the researcher tends to gain greater understanding of the project being researched as the research progresses. This allows the researcher to direct and redirect the questions to get data that answers the research questions.

By definition, a research method is a strategy of inquiry that moves from the underlying philosophical assumptions to research design and data collection. It

may be seen as a "general way of thinking about conducting qualitative research" Trochim (n.d.). The choice of research method influences the way in which the researcher collects data, analyses it and the subsequent presentation of the results. As Myers (1997) contends, specific research methods also imply different skills, assumptions and research practices.

Research method used in this study

Grounded Theory Method (GTM) was used as the research method in this study. There are many different ways of conducting research using Grounded Theory Method. Some of these are very prescriptive (Strauss and Corbin, 1990) but others leave room for the researcher to direct his or her research in the way that suits the research environment. The proponents of Grounded Theory Method, however, urge researchers to use the method flexibly (Glaser and Strauss, 1967; Charmaz, 2006). Charmaz (2006) also refuses to accept any prescriptive way of using this GTM. Instead she regards the method as a guiding framework, that is, as "a set of principles and practices" that any researcher can fine-tune to suit the context of research under study (Charmaz, 2006).

Grounded theory method (GTM)

Grounded Theory Method is a research method that seeks to develop theory that is grounded in data. According to Olivier (2004), Grounded Theory Method starts by observing the field of interest and theory is allowed to emerge from (is grounded) what is observed in the data. The data are systematically gathered and analyzed.

In addition, Trochim (n.d.) regards Grounded Theory Method as a generative method in that its purpose is to generate or produce theory. Cornford and Smithson (1996) regard GTM as a method that can be used to develop (induce) the final hypotheses, propositions, themes and classifications from the data that are gathered and analyzed as the study progresses.

The main idea of GTM is supported by Hacking (2002), who, in his discussion of the "Creation of Phenomena", asked the question, "What comes first, theory or experiment?" He contended that Hall, the discoverer of the Hall effect in magnetism, unequivocally supported the notion that experimentation should be the beginning of a theory. For anything to exist, it must be created. This inductive stance to theory development is dichotomous to the view that is followed when using deductive research approaches.

In terms of research design, Glaser and Strauss (1967) urge grounded method theorists not to start with a problem statement or research questions, but just an interest in the field will suffice. Adherence to this requirement influences the way a grounded researcher plans and executes the research study as will be described below. Contrary to this dictum, Strauss and Corbin (1990) advise researchers to have a preliminary hypothesis that should be used to guide the scope of the study.

The research design

Trochim (n.d.) regards research design as a process or phase that fastens the project components together. Each research activity is positioned and described according to its contribution to the overall goal of the research. It is a process of moving from methodological abstractions to descriptions of the practical steps that are proposed and later followed during execution of the research study. Since our chosen research method is the Grounded Theory Method, the research design should meet the requirements for its use in this research.

The research started with generic research questions that were proposed only as to guide and direct the scope of

the research (Mavetera and Kroeze, 2009). These research questions followed Roode (1993)'s process-based research framework that looks at the what, why, how and how should, of software development practices.

The data for this research study were gathered through the use of unstructured open interviews. The respondents were software development practitioners in South Africa. The interviews were recorded on tape and later transcribed with the help of professional transcription services. After that, the interview data was loaded as primary documents for coding into Atlas.Ti, a qualitative data analysis tool. The first three interviews were used for open coding, the remainder being used for selective coding and for reaching theoretical saturation. It is important to note that the process of literature study was done continuously and in parallel with the data-gathering and data-analysis processes. This literature study increased the theoretical sensitivity of the researcher. The following section discusses some software development issues that are documented in software development literature books as affecting the development of adaptive software products.

Software development issues

The complex nature of organizational systems

Organizational systems are examples of dynamic and complex systems. The complexity of these systems can be measured using the concept of requisite variety as proposed by Rosenkranz and Holten (2007). Requisite variety views organizational systems as possessing several possible states, in terms of "*patterns of behavior*" or "*number of manifestations".* During software development, it is the developers' intention to capture and maintain these patterns of behavior (manifestations) in the resultant software product. However, during the software development process, the tasks of modeling, that is, of developing the analysis, design and implementation models, tend to reduce the complexity of these organizational systems by reducing their requisite variety. This in turn reduces the possible behavioral states of the subsequent software products and information systems under development. This process is regarded as the reductionist principle. Any reduction in the organizations possible behavioral states and, hence, in their requisite variety, reduces in turn the life responsiveness of the modeled and developed system.

In order to maintain the requisite variety of organizational systems and to transfer them to the developed systems, either the modeled system must have its requisite variety reconstituted to the organizational systems' originally unmodelled state or the original system should not be modeled using the reductionist principle. Another option would be to allow the implementing tools used in the system and the users of the system to have as much of the requisite variety as the original unmodelled system possessed (Rosenkranz and Holten, 2007). Practically, it is impossible to have tools and users that have the same behavioral modes as the original system. In the end, the only practical way to do this is to find methods and tools that maintain the requisite variety during the process of system development. The requisite variety of organizational systems is observed in the culture, context and the concepts used in the organization.

Culture and concepts in organizations

Another aspect that contributes to the requisite variety of organizational systems is culture. Organizational culture comprises people's attitudes and experiences, as well as the beliefs and values of an organization. It also embodies the organization's interactional behavior with stakeholders. All organizations are run within certain cultural boundaries. Organizational culture is therefore reflected in the

concepts that are used in these organizations.

The success of a development team to build a language community with all the stakeholders in the development process and, hence, communicating concepts in the domain effectively, is regarded as one of the success metrics of software development.

Dahlbom and Mathiassen (1993) agree with Buitelaar et al. (2005) that concepts are not packages of information defined by structured complexes of more elementary concepts. They argue that concepts are defined by people's practices (Buitelaar et al, 2005). These concepts should satisfy three basic things, that is, they should have an intensional aspect, a set of the concept's instances (its extension) and a set of linguistic realizations. Linguistic realizations refer to the multilingual terms that are applied to the concept (Buitelaar et al, 2005). The use of concepts in a practice defines the organizational culture. Together they can be used to define the organizational context.

Context in organizational systems

As discussed above, concepts apply to a specific practice within a certain context. Roque et al (2003) emphasize the need for context to be understood if one is to successfully develop a software artifact. The software artifact should be fashioned so that it fits the context in which it is to be used. It must have situatedness.

Organizational communication requires a language to pass information between actors. To enable the actors to understand each other, this language should be "*embedded within the context*" of the organization (Malinowski, 1923). Messages that are communicated in an organizational system are also situated in that particular context. As such, information systems should be tuned in such a way that they capture this organizational context. Capturing organizational context subsequently captures meaning, a very important facet that actors need if they are to communicate. Context building amongst organizational actors is a process of weaving different situational understandings from different actors, of establishing threads of common understandings and the inter-subjective knowledge within the network 9Goldkuhl, 2002; Dilley, 1999). It must be noted that context is not a static thing.

It is not self evident in a situation but requires a constructive machinery to mould the varying situational meanings into a common understanding. Context therefore is an object of study that requires some analysis to have an agreed and shared understanding. It is the environment surrounding the meaning of a situation. At the same time, it is within the shared meaning of some situatedness that the said context resides. The next section is a discussion on other issues that affect the development of adaptive software products.

The development problem

Other issues which need consideration during software development range from focusing on the design of reusable components to focusing on the innovative elements of software product design. These innovative elements of a software product represent the domain-related additions that mark the difference between different types of domain packages.

With regard to domain, De Oliveira et al (2006) regard the lack of domain knowledge by software developers as the biggest problem in the development process. While the process of requirements elicitation and knowledge gathering are very laborious, knowledge sharing and reuse is also very limited. In these processes, for different software development projects, though in the same domain, one needs to explain the same concepts repetitively to different software development personnel. In

addition, developers have to study and learn the domain while simultaneously linking them to the tasks to which they relate. These tasks pertain to the problem domain that needs to be addressed by the subsequent software product (De Oliveira et al., 2006). Another issue that needs to be addressed in software development is the issue of communication.

Communication in software development

In all development processes, the analysis phase is tasked with creating a true reflection of the organizational system environment. As such, the analysis phase should result in the development of an analysis model that is both descriptive and in the form of a computationally independent model (CIM) (Aβmann et al. (2006). The CIM possesses as little platform information as possible, at the same time ensuring that the customer's viewpoint is maintained. The computationally independent viewpoint captures the system environment and the system requirements. However, many techniques have been used in traditional analysis modeling that ensures that this analysis model is "*expressed in terms of the problem domain*". Figure 1 depicts the intended relationship in communication that should exist between the analysis model, design model and the implementation model during their development

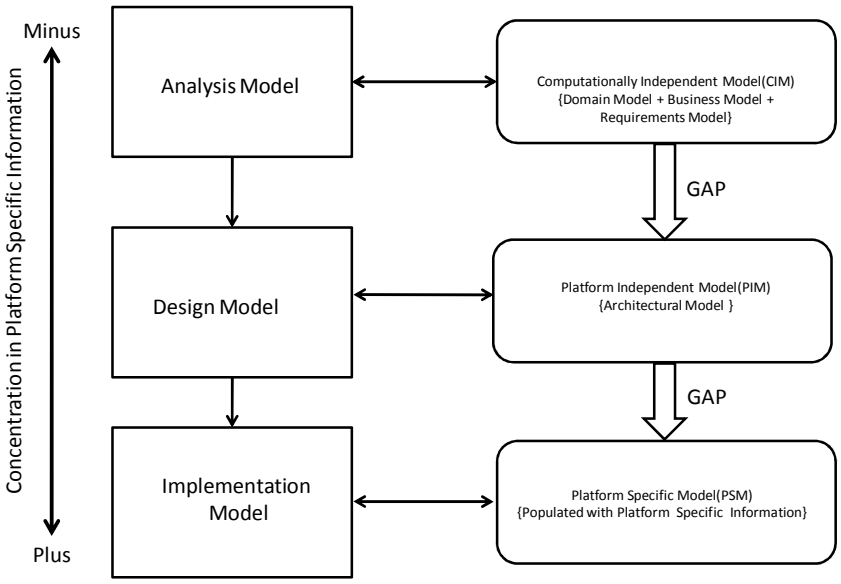

Figure 1 Communication Gap in Software Development

As discussed earlier, the analysis model is derived from the organizational system's environment. This environment is characterized by the domain and business information, both of which are context related. In addition, the system requirements, which derive their fit from the organizational context, are added. The analysis model is intended to capture the triplet of domain model, business model and requirements model. As is common, the requirements model manifests itself as the system specification. The design model is the architectural model of the system and, at this stage, it should capture the system from the designers' viewpoint while at the same time maintaining its platform independency. Lastly, the implementation model derived from the

design model is gradually populated with platform specific details as discussed by Aβmann et al. (2006).

It must be noted that the domain model is a product of the process of domain engineering. As Bjorner (2008) submits, understanding and capturing human behavior are the prime purposes of this process. Organizational human behavior has to be described formally or informally and communicated throughout all the software developmental stages. This same domain model captures concepts used in the domain field as well as the relationships between these concepts. The business model is tasked with capturing the company's rules of business, while the requirements model, which models the system specification, is tasked with capturing the functional and non-functional system requirements (Aβmann et al. 2006). Most of the time, system requirements refer to program code, hence implying that the specifications are platform dependent. For specifications to be platform independent, some informal specification methods should be used. Bjorner (2008) argues that this may require the specification to be drawn up in a natural language of some sort. This has not been the case with all specifications especially when developers get to the later stages of development. This Bjorner (2008) decries and submits that '*there are very many aspects of requirements that we today, 2008, do not know how to capture formally…".* As a result, many design specifications do not represent the true world view of a system.

As is always evident in software development, developers' insistency on formalization results in the development of mechanistic systems. In these mechanistic development methods, from the analysis model through to the implementation model, platform-specific information is allowed to creep into the system as a result of this formalization. The problem faced by software developers is that of translating all the characteristics of the analysis model (CIM) to the implementation model through the design model (PIM). This is normally because, at the end of the analysis stage, the system requirements are translated to a specification model (SM). It must be noted that this SM is an instantiation of parts of the functions of the system. This is a persistent problem in software development and ways of addressing it are proving to be elusive.

As stipulated by Aβmann et al. (2006) '*a specification model is a prescriptive model, representing a set of artifacts by a set of concepts, their interrelations, and constraints under the closed world assumption'.* The failure of the SM to transfer descriptive information captured by the analysis model to subsequent stages poses a serious problem in software development. The problems of capturing these issues during software development and of maintaining the organizational context in the software product were included in the data-gathering interview questions. The data gathering and its subsequent analysis revealed a plethora of issues that are discussed below as 'practitioner's perspectives' to software development and methods.

Practitioners' perspectives

Figure 2 depicts four major areas that were identified by respondents as warranting attention. These therefore need to be addressed in any software development process. As shown in Figure 2, the respondents identified the world view, the development paradigm, the research approach and the research methods as the four areas that need consideration when software products are being developed. They also identified two world views that are dichotomous and should also be considered when software products are being developed.

These world views, the mechanistic and romantic world views should therefore be afforded the highest priority (Priority I in Figure 2*)* in the development of an approach to software development. The romantic world view posits reality as a

social construction (Struwig and Stead, 2004) and believes in a shared reality among actors. Most importantly, what may be considered as knowledge depends heavily on the context, as well as on organizational politics and culture.

On the other hand, the mechanistic world view conceives reality as existing and as a given (Struwig and Stead, 2004). Using the results from axial coding, the mechanistic world view in contrast to the romantic world view is considered to be very syntactic.

As shown in Figure 2, every software development paradigm is associated with a world view. The second priority level (Priority II) therefore depicts the relationship between the world view, the development paradigm and the software development approach. The interviewed practitioners contended that a software development paradigm must be related and derived from a world view at the same time it must lead to the development of a software development approach. Therefore, two dichotomous software development paradigms: the realistic and the relativistic paradigms are noted. The choice of a software development paradigm was raised by the respondents as a fundamental issue when software products are being developed. They noted that many software products are delivered within an ill-founded grounding, that is, the development paradigm.

Furthermore, all the respondents concurred that, although there is an association between the development approach, the development paradigm adopted and the development method, development approaches should lie in between the paradigm and the method. The development approaches should derive and share the same philosophical underpinnings as the paradigm. The study therefore revealed an existence of three classes of development approaches: the structural approach, the behavioral approach and *approaches in transition.* Approaches in transition lie between the structured and behavioral approaches.

The class of structured approaches, usually referred to as traditional approaches, include the classical structured approach and the object-oriented approaches. At the other pole, we find the behavioral approaches, which encompass Checkland's widely read (but rarely used) soft systems approach. Behavioral approaches from which behavioral methodologies are derived assume a holistic organizational perspective (Benson and Standing, 2005). They accept the social construction nature of software products as well as the information systems they implement and therefore should assume a relativistic paradigm and a romantic world view.

In the middle, the approaches in transition, we find approaches that exhibit both the syntactic characteristics of traditional structured and the softer humanist elements heavily embedded in the soft systems approaches. This leads Brown et al. (2004) to refer to agile approaches as neo-humanist approaches. This is in recognition of the fact that many agile methods are touted to include techniques that capture the human aspects of organizations during analysis although they become very mechanistic at the design and implementation phases. This therefore qualifies agile approaches for inclusion in the approaches in transition. As the third level cannot be achieved without considering the world view and the development paradigm, we allocate it priority level III as indicated in Figure 2.

Lastly, the development method lies at the bottom of Figure 2. The reason for this stems from the fact that an approach dictates the group of methods that will be used at lower levels to develop software products. An approach which can erroneously be likened to a methodology is referred to as a study of methods. These methods are a way of selecting and using specific techniques and tools to accomplish a software

development project Bjorner (2008). The methods are therefore at the lower, more prescriptive phase of software development phases. There are however many methods that are considered and used in software development of which only communication methods will be discussed here as it raised a lot of interest from the interview respondents. In discussing development methods, the respondents gave the highest priority to the need for communication methods that can transfer the business model through all the development stages from analysis to implementation.

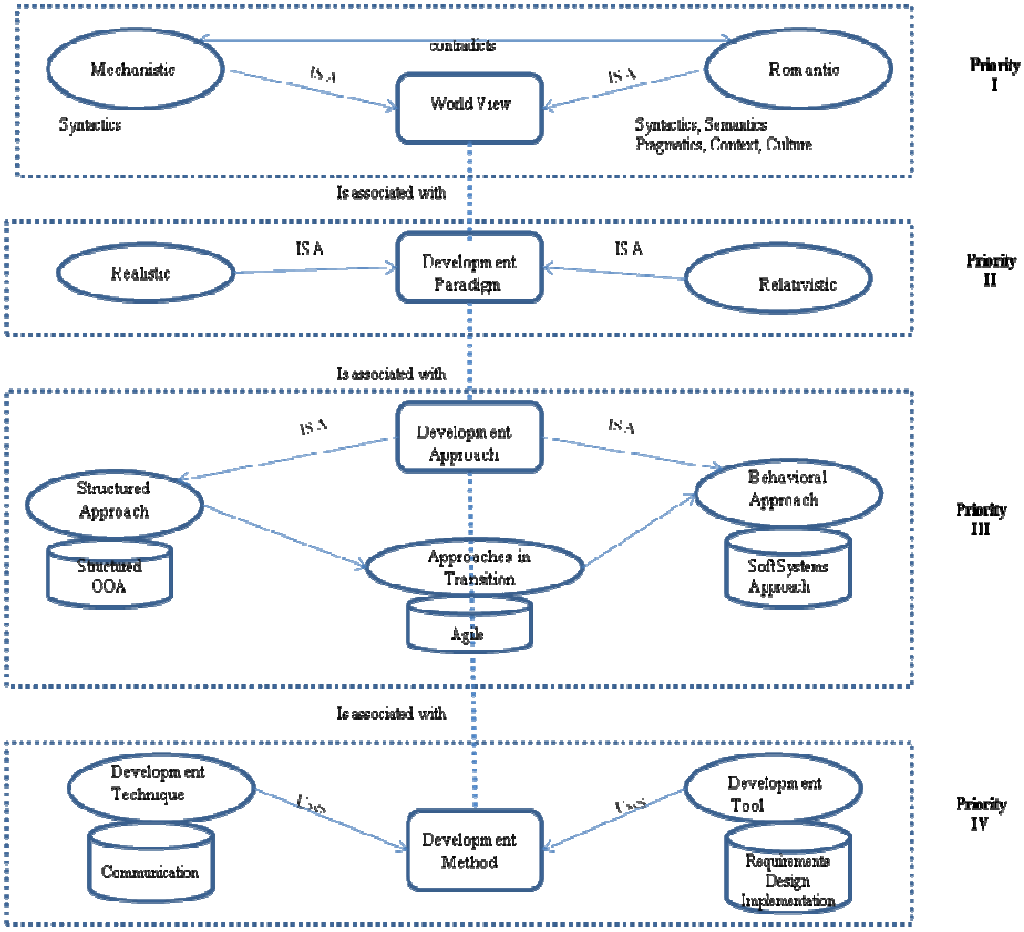

Figure 2: Software development Approach Components

The Research Propositions

The study has opted to use propositions instead of hypotheses strictly because unlike the former, the later has to be stated in a testable form, which allows the relationship between two or more variables to be examined. In many cases, a hypothesis is a suggested solution to a problem. This study is making empirical generalisations from the facts collected from software practitioners. These generalisations are therefore regarded as propositions, and are derived as a result of an exercise in deduction after observing themes and relationships in an empirical situation.

Based on the open coding and axial coding results of the first three interview data samples and closely related to Figure 2, the following propositions that

need to be considered during software development were made. From the analysis, it was discovered that:

Proposition (P1): *The field of software development lacks the correct paradigm, that is, it must therefore be positioned in the relativistic world view instead of the current realistic world view.*

Proposition (P2): *The software product development process requires an approach that will ensure the capturing of soft human elements or behavioral states that are inherent in organizational systems.*

Proposition (P3a): *The software development process requires communication methods that will ensure that all the stakeholders in the development process understand each other.*

Proposition (P3b): *The field of software development requires a method or tool that will capture the organizational context during analysis and maintain it through the subsequent development stages and the development life cycle of the system.*

Proposition (P4): *There are no proper methods that can be used to develop adaptive or evolvable software products that characterize organizations as dynamic systems. A method is therefore required to dynamically represent an organizational system as a dynamic piece of software.*

Proposition (P5): *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 descriptive analysis model characteristics to the design and implementation models without losing the informal domain- and business-related requirements.*

These **Proposition**s are addressed using elements from the software development framework discussed below.

The adaptive software development framework

The propositions made above are supported by a list of software development requirements that vary according to the nature of the software product under development, the phase of the software development life cycle and the general environmental characteristics of the development platform. In order to develop adaptive software products, the following requirements are mooted in a software development approach:

Proposition (P1) 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 developed.

Proposition (P2) requirements:

- The software development process should capture the softer issues of organizations, together with their organizational behavior.
- The software development process should reduce the dominance of traditional approaches and move towards a behavioral type of development approach.
- There should be a switch from the hard systems paradigm to soft system approaches.
- The approach should ensure a transition from a task-based approach to a role-based approach.
- The methodological approach should have a way of capturing

the dynamic nature of the ever-running context in organizations.

- The approach should ensure that software developers are able to study the organizational environment, live in it so that they can have a situated practice and experience this practice before they embark on any software development project.
- The functional requirements and the system requirements should be mapped from the organizational environment to the systems platform through a software model that does not overlook the social or human aspects of the organizational system.
- In addition to being able to modeling behavior, the organizational culture and context should also be captured.
- The software development process should capture and maintain the patterns of behavior of organizational systems

Propositions (P3a and b) requirements:

- The software development process requires a method that captures the ever-running organizational context and application domain of the system.
- The software development process requires a concept negotiation technique.
- The software development process requires a knowledge-sharing platform for all stakeholders
- A method is required that captures the semantics of the system.
- 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.
- The software development process requires a language that will capture human behavioral characteristics during development and allow their transfer or sharing among stakeholders.
- In current modern and pervasive computing environments which software development is outsourced offshore, the development approach should have a platform to enable different developers to share their understanding of the system requirements.
- Developers should be able to build a language community with all stakeholders, that is, there should be a linguistic model that could be used to negotiate a shared understanding of the concepts found in a system. This requirement supports the need for improved communication methods, techniques and tools that can be used during the development process.
- The software development process should have a developer as a tool that reduces the communication gap between systems analysts and the users. This would enhance user requirements gathering and their faithful transfer to the analysis model.
- Plain language, understood by all the stakeholders should be used during communication when doing requirements gathering. It is also important to include a business analyst, a person with business orientation to do the analysis. This is like capturing the domain and business model of the system. The systems analysts can then be incorporated to capture the system requirements.

- There is much mistrust between developers and users. This lack of trust leads to poor communication, resulting in poor requirements gathering. There should be a tool that captures or negotiates between users and developers.

Proposition (P4) requirements:

- 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 fast and easy.
- The software development process should enable evolvable software products to be developed.
- There should be a method of dynamically representing systems as dynamic pieces of software.
- There should be a method of capturing and modeling the design phase elements of a system in order to enable all the intended functions of the proposed system to be captured.
- An analysis model, derived from the domain theory should be designed for a family of systems in the same domain.
- The software product should be able to capture semantic and pragmatic (tacit) information in organizations.

Proposition (P5) 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 into the software product.
- Language limitation is the factor that most inhibits and limits the ability of software products to capture informal requirements in systems. A language is needed that allows the building of a language community and facilitate knowledge sharing using concepts.
- A development tool is required that captures the analysis model and must have a requirements repository that can capture and store user requirements during analysis.
- This repository should be able to be consulted at every stage of the software development life cycle of the system. Besides improving on requirements communication throughout the project, this would also increase the time to market and the quality of the software product.
- A methodology dictates the way a software engineering environment is subsequently used. It must be noted that, introduction of software engineering environment on its own could cause problems of fit. Since user requirements gathering takes up 80% of development time, it is important to have a development tool that speeds up the process. Without this, developers run the risk of rushing the requirements-gathering process and of implementing an incorrect system. Therefore, a software engineering environment is required to accompany ant software development approach or methodology that is used.
- There should be a way of reusing domain knowledge in software development, as discussed by De Oliveira et al. (2006).

- As De Oliveira et al. (2006) explains, 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 of the organization. This knowledge sets the context of the software product.
- There is a requirement for checking the quality of the software product throughout the development process.
- The software development process should allow for the capturing, storage and maintenance of the organizational business model throughout the development stages.

These propositions while many researchers and authors have written a lot on them, they still remain persistent and solutions to them are increasingly becoming elusive. It is therefore paramount to explicitly devote a paper on their nature and what is needed in software practice if they are to be addressed.

Conclusion and discussions

The development of adaptive software products has been hampered by many factors. As discussed in this paper, capturing the culture and organizational context are among the major issues that may improve the software development process. In order to develop adaptive software products, the propositions discussed in this paper, have to be accepted, adopted and remedies found.

As it is difficult to find a development approach that addresses all the requirements listed in these propositions, a gradual approach to developing methods, techniques and tools that can be used to develop adaptive software products should be followed. Future research looks at developing a software development methodology that incorporates most if not all of the propositions listed herein. In addition, the techniques and tools required for one to use the proposed methodology must be developed as well. In conclusion, it must be noted that, the process of software development has been made difficult by the failure of developers to capture the continuous and ever-running context of organizational systems.

Acknowledgements

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

References

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

Benson, S. and Standing, C. (2005) Information Systems: A business approach, 2nd Ed. Wiley and Sons, Austarlia.

Bjorner, D. (2008) Software Engineering: An Unended Quest. A Doctor of Technology Thesis. Technology University of Denmark...

Brown, R., Nerur, S. and Slinkman, C. (2004) "The philosophical Shifts in Software Development," Proceedings *of the Tenth Americas Conference on Information Systems,* New York, August 2004.

Buitelaar, P., Cimiano, P. and Magnini, B. 2003. Ontology learning from the text: Overview. In: Ontology learning from text: Methods, Evaluations and Applications (Buitelaar, P. Cimiano, P. and Magnini, B., Eds), IOS Press, 2005.

Charmaz, K. Constructing Grounded Theory: A Practical Guide Through Qualitative Analysis, Sage, 2006.

Cornford, T. and Smithson, S. Project research in Information Systems: A Student Guide. Palgrave, New York, 1996.

Dahlbom, B. and Mathiassen, L. (1993) Computers in Context. The Philosophy and Practice of Systems Design, Chapter 2. Oxford: NCC Blackwell.

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

Dilley, P. (1999) "Queer theory: Under construction". *QSE: International Journal of Qualitative Studies in Education*, (*12*:5), 457-472.

Glaser, B. G. and Strauss, A. L. (1967) The Discovery of Grounded Theory: Strategies for Qualitative Research, Aldine De Gruyter, New York.

Glaser, B.G. (1992) Basics of Grounded Theory Analysis: Emergence vs. Forcing, Mill Valley, CA, Sociology Press.

Goldkuhl, G. (2002) "Anchoring Scientific Abstractions: Ontological and Linguistic Determination Following Socio-Instrumental Pragmatism". *Proceedings of the European Conference on Research Methods in Business and Management Studies (ECRM 2002),* MCIL, Reading, UK.

Gonzalez-Perez, C. and Henderson-Sellers, B. (2006) "An ontology for software development endeavors". In: *Ontologies for Software Engineering and Software Technology* (Coral Calero, Fransisco, Francisco Ruiz and Mario Piattini Eds), Springer Verlag.

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

Malinowski, B. (1923) The Problem of Meaning in Primitive Languages. *The Meaning of Meaning*. C.K. Ogden and I.A. Richards, Routledge and Keagan Paul, London, 296-346.

Mavetera, N. and Kroeze, J. (2009) "Practical considerations in Grounded Theory Method Research". *Sprouts: Working Papers on Information Systems*, (9: 32), [Online]. [Retrieved July 10, 2009], http://sprouts.aisnet.org/9-32, 2009. ISSN 1535-6078.

Myers, M. D. (1997) "Qualitative Research in Information Systems," *MIS Quarterly* Vol.21, No. 2, June 1997, pp. 241-242. MISQ Discovery, archival version, [Online], [Retrieved April 9, 2008], http://www.misq.org/discovery/MISQD_isworld

Olivier, S.M. (2004) Information Technology Research. A Practical Guide for Computer Science and Informatics. 2nd ed., Van Schaik, Pretoria, South Africa, SA.

Roode, J.D. (1993) Implications for Teaching of a Process Based Research Framework for Information Systems. Department of Informatics, University of Pretoria, 0002, Pretoria, South Africa.

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

Rosenkranz, C. and Holten, R. (2007), "Towards measuring the complexity of Information Systems: A language Critique Approach." Proceedings of

International Resources Management Association (IRMA) 2007, ISBN: 978-159904930-4.) 19-23 May 2007, Vancouver, British Columbia, Canada, 57-60.

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

Struwig, F.W. and Stead, G.B. (2004), Planning, designing and reporting research, Pearson Eduaction, Cape Town, South Africa.

Trochim, W. M., n.d. The Research Methods Knowledge Base, 2nd Edition. [Online], [Retrieved, 7 April 2008], http://www.socialresearchmethods.net/kb/.