Issues in Software Development Practices

A South African Software Practitioners' Viewpoint

Nehemiah Mavetera, North West University, Mafikeng, South Africa, Nehemiah.Mavetera@nwu.ac.za
Jan Kroeze, University of Pretoria, Pretoria, South Africa, Jan.Kroeze@up.ac.za

Abstract

Software development is a process tasked with the development of artefacts that are used to implement organizational information systems. Depending on the social, economical and environmental setting, different software practices are used. These, however, have an effect on the resultant software product.

In this paper, the authors investigate some of the software development practices that are used in South Africa. Through the use of interview techniques, the study highlighted a plethora of methods, techniques and tools that are used during the software development process.

This paper advocates for a paradigm shift in the way information systems are developed. It motivates for developers to consider the social context of organizational information systems when developing software products. In a social context, capturing the organizational culture, context and human aspect contributes to the system's responsiveness and its adaptiveness to the ever changing organizational environment.

Keywords: Grounded Theory Method, mechanistic system, software development, actor network theory

Introduction

The software development process is influenced by many factors: paradigmatic, methodological, technological, human and, most importantly, organizational issues. These factors at some point or other influence the usability of the software product. Kawalek and Leonard (1996) discuss a paradox in which software products are regarded as a hindrance to organizational change and progress. One reason for this is the fact that a holistic real world system is forcibly embedded in a piece of software as a "representation of an organization". Using Kant's Philosophy of deconstruction (Gasche, 1986), the piece of software is just a partially understood way to the organizational system at any point in time. Unlike software products, organizations are always in a state of flux (Dahlbom and Mathiassen, 1993; Kawalek and Leonard, 1996) and change and duplicate themselves in their existing environments. Developers need to match the static nature of software products and the dynamic nature of the organization to the dynamic nature of the software model.

This recognizes two paradigms in software development, that is the current mechanistic practice and the romantic paradigm. The software development paradigms, some approaches and methodologies will be covered in the discussion. This discussion will be used to identify the issues that contributed to the development of currently available mechanistic software products.

The structure of the paper is as follows: Section 2 gives a brief background of the research problem. Section 3 discusses software development practices, organizations and software products as representations of organizational information systems. Section 4 discusses three theoretical frameworks: the Activity Theory (AT), Actor-Network Theory (ANT) and the Theory of Organized Activity (TOA) and how they are used to explain the relationship between the actors in an organisational information system. The research design, based on a qualitative field study is described in Section 5 and the issues affecting software development in South Africa are covered in Section 6. Section 7 of the paper makes some recommendations regarding the factors that should be considered if the software development process is to be improved.

The Problem

The nature of the software product is influenced by the real world artefact it must represent. As software products are derived from a software model, this software model should be a true representation of an organizational system. Kawalek and Leonard (1996) say that, unlike

software artefacts, organizations are always in a continuous state of change. This means that it is difficult for software products to represent the true state of an organization at any particular time. Kawalek and Leonard (1996) and Meso and Jain (2006) state that software products representing an ever-changing organisation should be innovative, adaptive and replicable. Thus new methods of developing software products that are adaptive and can evolve continuously as the organisation changes should be formulated.

Kawalek and Leonard (1996) and Lehman (1991) argue for both a technological and methodological evolutionary viewpoint when software products are developed. This viewpoint can ensure a synergic relationship between the organization and the subsystems that are implemented using software products. Evolving systems should incorporate the organizational context, which is also dynamic. The inclusion of context in the software design base renders the software product development process uncertain. On the other hand, software development processes have always concentrated on the static (S-type), that is, on the correctness of the software product design being regarded as the only criterion for software product success, coupled with its fidelity "in a strictly mathematical sense" to the specification.

Reliance on the S-type paradigm has had an effect on the conformity of the software product to the real world, since software products cannot move from one steady state to another steady state if they are to implement what occurs in a dynamic organisation. In other words, software products cannot be optimised against static goals (Kawalek and Leonard, 1996).

Kawalek and Leonard (1996) lamented the failure to develop methods and practices that produce "instantly adaptable software that is able to support radically changing demands on a series of fast developing platforms and integrating with a series of end user developments". In short, organisational context is always changing, but the software specification is a static photo reflection of the time it is signed off, which is later mapped on to the static software product. This notion has led to the problem of mapping a world view to a conceptual view and, lastly, mapping of the conceptual view to the logical and physical view of the software product. Section 3 discusses some of the software development practices are currently in use in industry.

Software Development Practices

The practice of software development has mainly been principally guided by the definition given to software engineering. Several authors (Schach, 2005; Pressman, 2005; Heineman, 2000 and IEEE) have given software engineering (SE) a plethora of meanings but all of them borrow their core meaning from the definition of the term "engineering".

All the definitions emphasize issues that are mechanistically oriented, such as use of the word design (ECPD), systematicity, quantification and engineering of software (IEEE), fault-free, delivered on time and within budget (Schach, 2005). Although Schach (2005) mentions the need to satisfy users' needs and Pressman (2005) refers to a set of methods, the inclusion of the social and, hence, of the contextual human issues that are considered in the definitions is not reflected in their statements.

Schach (2005) suggests that the practice of software engineering should incorporate disciplines such as sociology, philosophy, management and psychology, in addition to other highly formalized disciplines such as mathematics, computer science and economics, to name but a few.

As stated earlier, software development is an attempt to map the software product to an organizational system. The inclusion of people makes organizational systems both very dynamic and very complex. Organizational complexity is measured using the concept of requisite variety (Rosenkranz and Holten, 2007). Requisite variety views information systems and organizations as possessing several possible states in terms of "patterns of behaviour" or "number of manifestations". During software development, it is the developers' intention to maintain these patterns of behaviour (manifestations) in the resultant software product. On the other hand, the current software practices canvassed in the development philosophies of structured, object-oriented and agile methodologies tend to reduce the complexity of these information systems by reducing their requisite variety.

Structured, object-oriented and agile development philosophies are guided by the

principles of systematicity, system formation and deconstruction (Gasche, 1986), explicit programming (Agentis, n.d.) and reductionism, all of which have their grounding in the functionalist paradigm of Burrell and Morgan, (1979). The functionalist paradigm, in addition to idealizing the real world, also enforces Aristotle's dictum that the whole is equal to the sum of its parts, a notion that does not apply to a general system such as an organization.

For the sake of completeness, some of the terms used above are described and explained in this paragraph. Systematicity refers to the extent to which a system can be regarded as an ordered, hierarchical arrangement of components. System formation is concerned with the process of building up the whole system from its components. The system building process has to be ordered and well organized. This explains Kant's philosophy of deconstruction, that is, finding the extent to which the principles of systematicity and system formation can be applied during the reductionist processes of structurally breaking up the system features during analysis, design and later re-grouping (re-assembling) them at implementation stage to recreate the whole, the general system.

Coupled with the principle of requisite variety, the principle of deconstruction views the original system (general system) as "not the universal essence of systematicity; rather it represents the ordered cluster of traits of possibilities which in one and the same movement, constitute and deconstitute systems" (Gasche , 1986).

This leads us to conceive of information systems as artefacts made up of a continuous connection, a chained arrangement of its constitutive parts. These constitutive parts, once fragmented and reassembled, cannot re-create the original whole. This contradicts Aristotle's maxim stated above that the "whole is equal to the sum of its constitutive parts".

Although context determines meaning and meaning is a by-product of a social context and both are a continuum and temporal, information systems, as social constructions, cannot be constituted fully from the products of their deconstructions. In its totality, a general system reconstituted (developed) from the mechanistic principles of reductionism cannot have unity of purpose or focus or a horizon of meaning, sense or context which gives it the attributes of a total or a whole system (Gasche, 1986).

Although systematicity and system-formation through the application of reductionism have been used to construct general systems (Gasche , 1986), the system components could not be reunited into "one well rounded-off system". Reductionism is the result of the successful idealization of information systems, although the resultant information systems lack idealization (Gasche , 1986). Reductionism reduces the possible behavioural states of the system under construction. As a result, the life responsiveness of the modelled and subsequent system is reduced.

In order to maintain the variety of the systems, either the modelled system should have its variety reconstituted to its original (unmodelled state) or the system should never be modelled according to the reductionist principle. Instead, the implementing tools, as well as the users, should possess as much variety as the original systems possessed (Rosenkranz and Holten, 2007). As the latter process is impossible to execute, the former process becomes the only practical way of achieving this.

3.1 Use of Automated Tools

As people develop systems, there is an over-reliance on the use of automated formal tools. These tools, such as modelling tools and code-generating tools, have limited the capacity to which developers can capture a system's behavioural characteristics (Lemmens, 2006). It should be noted that the automated tools are very good at task definition and task decomposition, tasks that are addressed at the design and implementation phases of the system development process, but which are very poor at maintaining the requisite variety of the original life system. The variety which is embedded in the behavioural aspects of the system can only be captured at the analysis stage. This process leads to the development of mechanistic systems.

The field of software engineering has largely been confined to the hard sciences discipline in which each software product is viewed as implementing a system with a goal, a definite boundary, and which is closed and deterministic.

This notion disregards the fact that the software product is not an end but a means to an end: the implementation of information systems. As a means to an end, the software product should take into cognizance the behavioural characteristics of information systems during its development. Information systems have multiple goals and are non-deterministic; their boundaries are not definitive but are permeable and open (Du Plooy, 2004). In fact, the discipline of software engineering should be regarded as a 'soft discipline' to which Checkland's (1999) soft systems thinking can be applied. In summary, automated tools are the products of an engineering process and if used as such, result in a reduction in the requisite variety of software products.

3.2 Software Products Component Reuse

Software practitioners are looking at methods that make use of already developed software products (component re-use). Current attempts are limited to the re-use of codes from component libraries, for example graphic user interface (GUI) designs. These methods, however, cannot be used to construct a complete application from entirely "pre-existing independently developed components" (Heineman, 2000). Heineman (2000) also makes a distinction between software evolution and software adaptation. Software evolution deals with the modification of software components, whereas in adaptation, the developer adapts the software product for a different application. To achieve software adaptation, developers need to know the specific configuration of a particular piece of software to be adapted. This requires knowledge about the product architecture, which is usually a privileged piece of information retained by the original manufacturer.

Most of the issues that impact negatively on the usability of software products are tied to the emphasis by software developers on tractability and objectivity during the process of identifying requirements. The field of software development has a social nature. Software requirements are a social construction in which human and non-human actors participate.

In short, the accepted norms in fashioning software products include the principle of deconstruction, systematicity and system formation, the reductionist principle and the dependency or over-reliance on functionalism to define the nature of the organisational system. On the other hand, the reality about organisational systems is that people are the most important element, that there exists a culture in these systems that needs to be captured and included in the IS and, lastly, that a specific context applies to each individual human activity system to be implemented using the software product.

Three theoretical frameworks that can be used to explain the social nature of information systems, organisations and hence the software products that represent them are discussed in Section 4.

Theoretical Grounding

Many of the software development practices discussed above rely on the principles of deconstruction, systematicity and system formation, as well as on the reductionist principles that are all dependent on the functionalist paradigm. In order to find the most appropriate software development practice, systems have to be evaluated using one or all of the following three theoretical frameworks, that is, the activity theory (AT), the actor network theory (ANT) and the theory of organized activity (TOA). Because of the inclusion of people as components of information systems, these frameworks accept the existence and role of culture, context and pragmatics in the fashioning of software products.

4.1 Activity Theory

Activity Theory (AT) framework has been used to explain and facilitate the understanding of human activity systems. An activity is the smallest indivisible, action-oriented and goal-directed process that can be found in a system. Taking an activity as a basic unit of analysis in organizations, it is used to explain the "coherence of individual actions in a larger social context" (Roque et al., 2003). In this regard, an information system thus consists of an assembly of individual activities that work together synergically to achieve an organizational goal.

These basic components of information systems need to communicate through a mediator, in order to satisfy organizational requirements. The various interacting components can be referred to as actors. The role of communication between the actors is carried by mediators.

These mediators are artefacts usually software products. These artefacts, in their role as

mediators, mediate between individual humans, between humans, between humans and technological artefacts (hardware and software) and between purely technological artefacts. This state of affairs leads to the understanding that mediator software products work in a socio-technical human activity system.

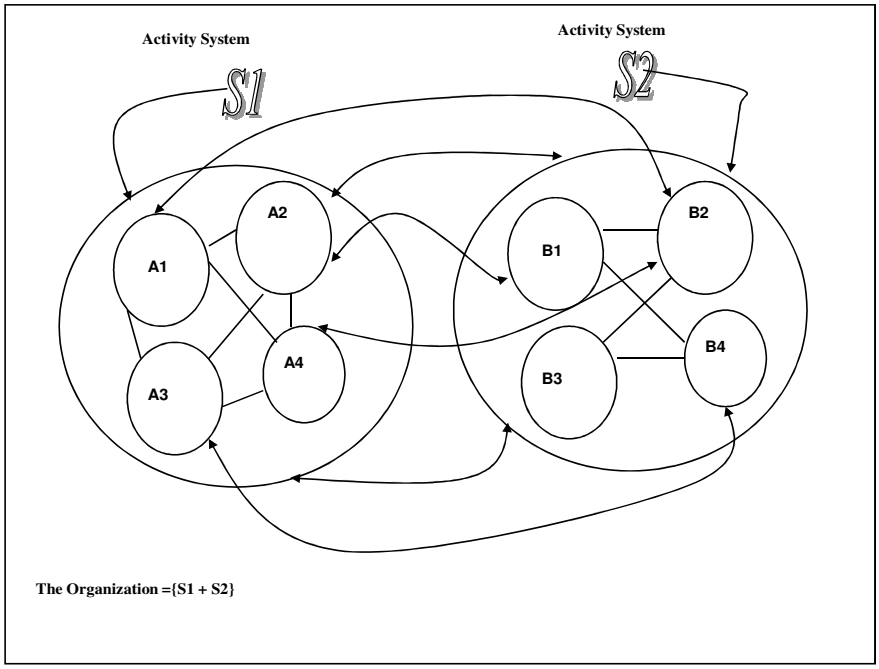

Figure 1 An Activity System: Adapted from Roque et al (2003)

In Figure 1, activities A1-A4 and B1-B4 are considered as individual activities. S1 and S2 are distinct activity systems as well. Activity system S1 is made up of actors A1-A4. Activity system S2 comprises actors B1-B4. Systems S1 + S2 together give rise to yet another combined activity system, i.e. the organization.

In practice, goals for activities A1 to A4 and B1 to B4 can be determined a priori but it is difficult to determine the goals which result from their interaction to make activity systems S1 and S2 respectively. More so, the combination of S1 and S2 to make "O", the whole organization becomes even more complicated.

It should be remembered that the most important element of any information system is 'people'. By assuming the Aristotelian principle, that is, that the sum of the parts is equal to the whole, information system development (ISD) approaches have failed to consider the human behavioural aspect of activity systems. In human psychology and behavioural sciences, the whole may not necessarily be equal to the sum of its parts. Determination of the nature of a process (activity) and its outcomes, while "ignoring changes in motives and goals, ignoring actors, human and non-human and ignoring the multiplicity of disciplinary agencies involved" is why information systems often fail (Roque et al., 2003). The software product, as the mediator, should be fashioned to accommodate all these factors.

4.2 Actor Network Theory

The actor network theory (ANT) can be used to explain and justify the social network aspect of human activity systems. Figure 1 shows that each activity systems A1 to A4, B1 to B4, i.e. groups S1 and S2 can again be referred to as actors. Within each activity, a software product mediates and creates communication channels, in turn making each activity system an actor-

network. In this case, we can regard the software product as an obligatory passage point (Introna, 1997) and as an artefact that is used to transfer messages from one point to the other, ensuring synergy in communication. Latour's (1999) actor-network theory (ANT) can be used here as a theoretical lens to explain the social interactions and relationships that exist between each actor in an activity system.

ANT regards each component of an activity system as an actor that is influenced by another actor through the mediation of other actors. These actors could either be human or technological. The actor as an "author of inscriptions is a network itself, and centre of translations. It is also influenced by relationships established by itself as a node in the network where other actors participate" (Roque et al., 2003).

To summarise, the actors participating in the activity system exhibit both some human voluntarism and technological determinism whose interplay results in the emergence of complex social characteristics. In the spirit of ANT, any actor, whether loaded with inscriptions or translations, or the mediator or mediated, should be fashioned in such a way that it is allowed to evolve and co-evolve with other actors. It must be allowed to adapt to the ever-changing requirements of the environment. All these responsibilities are given to the software product. It should also be emphasized that organizational systems are heterogeneous social entities that are always in a state requiring continuous maintenance.

4.3 Theory of Organised Activity

According to Holt (1997), the theory of organised activity (TOA) is based on human (organised) activities. Organised activity is a dependent variable of the social interaction of people in a particular setting. As applied to information systems (IS), Cordeiro and Filipe (n.d.) view the technical aspect of IS as playing a supporting role to the whole organisational human activity.

They also describe the human action, that is, the action performed by a human actor in an actor network consisting of interests and actors. The interests and actors are therefore responsible for the actions observed. It is impossible for technical machines to have interests. Since technical machines cannot have interests, they cannot be assigned any organisational responsibilities. The end result is that the technical aspects of IS cannot perform actions (Cordeiro and Filipe, n.d.). In short, although technological artefacts may be components of an organised activity, their failure to ascribe interests places them at a disadvantage in being assigned responsibilities for an action.

In conclusion, if the three theories are combined, the following can be shown:
With respect to AT, activity systems should have mediators which should ensure the coherence of actions in the larger organisational context. These mediators work in a socio-technical activity system.
ANT acknowledges that activity systems are social networks. In such actor networks of activity systems, alongside mechanistic technological determinism, there is some human voluntarism that needs to be translated, using software products as obligatory passage points.
The principle of social evolution should be given high credence. Adaptation to the ever-changing, dynamic environment should also not be neglected.
TOA posits organised activity as a dependent variable of the social interaction that occurs in actor networks of human activity systems.
TOA stresses the necessity for any organised activity to occur in a particular setting, i.e. in a particular context. In TOA, technological determinism only plays a supporting role in human activity.
Organised activity requires that human action be performed. In any human action, there are interests, which cannot be ascribed to technological artefacts but to humans. Hence, only humans can be given responsibilities in an activity system.

The important aspects of AT, ANT and TOA discussions are that software and system developers should find and use a methodology that accepts and captures the culture and context that exist in organisational information systems. This methodology should be grounded on the principle that only humans have interests and that, therefore, only humans can perform actions. Technological artefacts only play a supporting role in human actions.

The Research Design

The purpose of this study was to identify the issues that are taken into consideration in the development of software products. This research

study assumed a qualitative explorative field study approach. The research approach dictates how the step-by-step execution of a research project is carried out. The authors decided to use the grounded theory method (GTM) since they discovered that the formulation of a prescriptive hypothesis or research problem would restrict their findings. As a research method, GTM met the purpose and requirements of the study and the part played by the researchers in the investigation, as well as data collection and analysis.

GTM is a generative type of research method. As discussed by Glaser and Strauss (1967), Strauss and Corbin (1990), Charmaz (2006) and Olivier (2004), GTM starts by identifying an area of interest. Data are then gathered, and trends in the data which manifest themselves as incidents and categories of incidents are deduced.

Data were gathered through the use of open interviews. The target respondents were IT professionals (academics, software developers, analysts, designers, project managers, system users) who had been engaged in this field for at least five years. This was done to limit the number of responses that would be purely academic and not coupled to some industrial experience in the development and use of software products.

The first interviews were directed at generating categories of issues in software development. The respondents were chosen as they had a generalist type of knowledge in software development. Three of the respondents were academics with some industrial experience and one was a project manager and IT consultant. After generation of the categories the remaining interviewees were chosen from a random sample of developers, analysts, designers, users and project managers. The later interviews focused on consolidating the categories generated from the first data samples. From the sample interviews used for this analysis, three people, a developer, an analyst and a software tester were retained.

The interviews were transcribed by a professional transcriber. The analysis was done using Atlas.Ti5.2, a qualitative data analysis piece of software. Before the interviews as primary documents were loaded onto the Atlas.Ti system, the researchers replayed the interviews and checked the fidelity of the transcriptions. This process also helped the researchers to recreate the interview atmosphere and enhanced their understanding of the interview content. In the analysis of the data obtained from the interviews, quotes which revealed trends in the data were allocated codes which were then grouped into families. This process allowed the researchers to group together those trends which reflected similar issues that are encountered during software development. Any theoretical insights were also recorded as memos both on the Atlas.Ti system and in a note pad. These memos allowed the researchers to find links in the software development issues discovered. The software development issues that were raised by the respondents are discussed in Section 6.

Issues Affecting Software Development Practices

Table 1 shows the different categories of issues that were mentioned by the respondents as affecting the software development process. In the category section, respondents' views were analyzed and coded using a concept or group of concepts as they apply to software development. The right hand column in Table 1 shows the number of quotes that were identified in the respondents' data that refer to the category listed in the left hand column of the table.

Table 1: Software Development Issues	
Category (Code Family)	**N° of Quotations**
Communication Requirements	13
Communication Techniques	11
Development Issues	4
Development Approaches	2
Development Methods	2
Development Techniques	5
Development Tools	3
Interface Issues	2
Syntactic Issues	2
Semantic Issues	2
Pragmatic Issues	0
Contextual Issues	5
Adaptive Issues	2
Software Quality	1

The different categories shown in Table 1 are described in Section 6.1. These categories were further grouped into communication issues, development issues, semiotic issues, quality issues, adaptive and interface issues.

Communication Issues

Communication issues combine the communication requirements and communication techniques categories 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 world view requirements to the systems view holistically?'

The respondents cited several issues which prevail in South Africa. These communication issues emphasize the need for software developers to involve the user throughout the project and highlight the need for tools and techniques such as brainstorming sessions, mind mapping, printable white boards, pair programming, user stories and flyers if the communication problem is to be solved. The respondents 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 discussion of AT, ANT and TOA, the need for a mediator in the software development process and in the organizational system itself was emphasized.

Development Issues

Among development issues, we find a set of development approaches, methods, techniques and tools. Some of these, such as extreme programming, pair programming, are already in use but 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."

The above statement highlights the need for developers not to design software products using the engineering definitions and requirements used in other industries. Development methods should be interactive and should allow the development of adaptive products. The choice of a development approach was based on the schedule requirements of the project and not the quality and functionality of the product. However, agile methods though were the preferred choice of many respondents.

Semiotic Issues

Semiotic issues (Stamper, 1992 and Sowa, 2000) 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 researchers found that current development methods emphasize the mapping of the syntactic software model on implementation platforms, such as programming languages. One of the respondents said:

"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".

The interviews 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 researchers found that many practitioners want to develop adaptive and evolvable software products but that the current development methods and platforms did not support the need for this. 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.

Conclusion and Recommendations

This paper highlights some very important issues that are found in the field of software development. It is recommended that a softer approach to software development be adopted. Such an approach would embrace the socio-technical nature of information systems. It recognises the duality of organisational context with that of developing software products.

If it is accepted that information systems are a social construction, software development methods that recognise this aspect should be found and used. The prominence of agile methods and techniques in the responses suggest that agile methods are a closer approximation to the development methods that accept the social nature of organizational information systems.

Another aspect worth discussing is the issue of semantics, pragmatics and context in software development. These aspects were rarely mentioned by the respondents, suggesting that practitioners' focus on adaptive and evolvable software products is very limited. This trend seems to indicate that mechanistic products will continue to be fashioned even though they do not satisfy users' requirements.

The problem of software development should be approached from the humanist paradigm and not the current functionalist paradigm. Starting with the philosophical groundings in agile approaches, a development approach that takes into account the three semiotic levels of semantics, pragmatics and context should be developed. The field of software does not require a methodological shift but a paradigm shift (Checkland, 1999) that considers the romanticism inherent in organizational information systems.

Acknowledgement

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

1. References

Agentis. n.d. Simplifying the complexity of Application Development. Developing and deploying J2EE solutions with Agentis Adaptive Enterprise TM Solution Suite, Agentis International, Inc.

Burrell, G., and Morgan, G. *Sociological Paradigms and Organisational Analysis.* London: Heinemann, United Kingdom, 1979.

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

Checkland, P. Systems Thinking, Systems Practice: Includes a 30-year retrospective, Wiley, Chichester, United Kingdom, 1999.

Cordeiro, J and Filipe, J. n.d. "Language Action perspective. Organisational Semantics and the theory of organized activity. A Comparison", Retrieved 4 May 2008 from: //ltodi.est.ips.pt/jcordeiro/documents/demo2003 CordFil_Final.pdf.

Dahlbom, B. And Mathiassen, L. Computers in Context. The philosophy and Practice of Systems Design, Oxford: NCC Blackwell, United Kingdom, 1993.

Du Plooy, N.F. *Information Technology change management.* University of Pretoria, Pretoria. (class notes), South Africa, 2004.

Engineers Council for Professional Development (ECPD), Retrieved 11 January 2008, from: //www.abet.org/history.shtml.

Glaser, B. G., and Strauss, A. L. *The Discovery of Grounded Theory: strategies for qualitative research,* Aldine De Gruyter, New York, United States of America, 1967.

Gasche, R. "Infrastructures and Systematicity" in *Deconstruction and Philosophy: The Texts of Jacques Derrida*. John Sallis (eds), 1986. pp 3-20, Software Engineering Notes, (25:1), January 2000, pp. 55.

Heineman, G.T. A Model for designing Adaptable Software Components. ACM, SIGSOFT., 2000.

Holt, A. W. *Organized Activity and Its Support by Computer,* Kluwer Academic Publishers, Dordrecht, Netherlands, 1997.

Introna, L.D. *Management Information and Power*, Macmillan Press, Basingstoke, 1997.

Kawalek, P. and Leonard, J. "Evolutionary software development to support organizational and business change: a case study account." *Journal of Information Technology (*11: 3), September 1996, pp. 185-198. Palgrave, Macmillan.

Latour, B. *Pandora's hope: essays on the reality of science studies*. Cambridge, MA: Harvard University Press, United States of America, 1999.

Lemmens, R. Semantic Interoperability. PhD Dissertation, ITC, Netherlands. 2006. Retrieved August 30, 2008, from: //www.ncg.knaw.nl/publicaties/geodesy/pdf/63lemmens.pdf.

Meso, P. And Jain, R. "Agile software development: Adaptive Systems Principles and Best Practices". *Information Systems Management (*23:3), June 2006, pp 19-30.

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

Pressman, R.S. *Software Engineering: A Practitioner's Approach*, 6th Ed, McGraw–Hill, International Edition, 2005.

Roque, L., Almeida, A. and Figueiredo, A. D. "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.

Rosenkranz, C., and Holten, R. "Towards measuring the complexity of Information Systems: A language Critique Approach". In *Proceedings of International Resources Management Association (IRMA)*, 2007, pp. 57-60.

Schach, S. R. 2005. Object–Oriented and Classical Software Engineering. WCB/McGraw-Hill.

Sowa, F. J. "Ontology, Metadata and Semiotics. Conceptual structures" in *Logical, Linguistic and Computational Issues,* Ganter and Mineau, (eds). Springer-Verlag, Berlin, Germany, 2000, pp. 55-81 (*Lecture notes in AI #1867).*

Stamper, R. "Signs, Organisations, Norms and Information Systems", In *Proceedings of the 3rd Australian Conference on Information Systems*, Wollongong, Australia, 1992.

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

Copyright © 2009 by the International Business Information Management Association (IBIMA). All rights reserved. Authors retain copyright for their manuscripts and provide this journal with a publication permission agreement as a part of IBIMA copyright agreement. IBIMA may not necessarily agree with the content of the manuscript. The content and proofreading of this manuscript as well as any errors are the sole responsibility of its author(s). No part or all of this work should be copied or reproduced in digital, hard, or any other format for commercial use without written permission. To purchase reprints of this article please e-mail: admin@ibima.org.