Software Quality Management Supported by Software Agent Technology

R. C Nienaber and A. Barnard
University of South Africa, Pretoria, South Africa

nienarc@unisa.ac.za barnaa@unisa.ac.za

Abstract

Software technology and computing resources have evolved and developed considerably over the past years and may be considered as the backbone of many business ventures today. However, the software project management environment has also changed and is continuously evolving. Currently software projects are developed and deployed in distributed, pervasive and collaborative environments. This means that traditional software project management methods cannot, and do not, address the added complexities found in a pervasive, distributed global environment. Projects thus have a high rate of failure. More specifically, software projects often do not comply with the traditional standard measurements of success, namely time, cost and specifications. There is thus a need for new methods and measures to support software project management.

In this paper, software agent technology is explored as a potential tool for enhancing software project management practices in general. We propose and discuss a software agent framework, specifically to support software quality management. Although still in its initial phases, research indicates some promise in enabling software developers to meet market expectations and produce projects timeously, within budget and to users' satisfaction.

Keywords: Software Project Management, Software Agent Technology, Project Quality Management.

Introduction

Information Systems (IS) play a major role in today's daily business activities, ranging from small business operations to enterprise-wide operations throughout the worldwide business community. With the advent of the Internet and related global networking capabilities becoming more pervasive, cost-effective computing resources will continue to play a major role in improving organizational operations.

Yet, over the past two decades, software projects frequently failed to live up to user expectations, were commonly delivered late, and mostly ran over the set budget. The Standish Group (2000) studied 13,522 projects in a survey named EXTREME CHAOS. This study determined that 23 percent of the surveyed projects failed, 49 percent did not meet the requirements and only 28 percent succeeded. In March 2003 the group reported that success rates increased to a third of all projects, but time overruns increased to 82 percent, whilst only 52 percent of required and specified functions and features were included in the

final product. Software developers and managers are alerted to the fact that these issues have to be addressed in concrete terms. In particular Brooks (1987) listed the invisibility, complexity, conformity, and inflexibility of software as complicating factors in managing software projects. Initially, techniques utilized in traditional Project Management (PM) practices were applied to the development of software projects. However, standard PM methods seemed to lack the capacity to address the unique characteristics of the software development arena (Hughes & Cotterell, 2002).

This led to the development of S*oftware Project Management* (SPM) as an independent application area and field of study. SPM includes, amongst other things, the management of all issues involved in the development of a software project, namely scope and objective identification, planning, evaluation, project development approaches, software effort and cost estimation, activity planning, monitoring and control, risk management, resource allocation, as well as managing contracts, teams of people and quality.

The SPM environment is continuously changing as a result of globalization and advances in computing technology. This implies that the traditional single project, commonly executed at a single location, has evolved into distributed, collaborative projects. A number of emerging capabilities, e.g. agent technology and automation, network-centric operations (Durham & Torrez, 2004) and grid/distributed computing are providing a novel infrastructure for connecting otherwise isolated computing resources, and supporting the development and management of distributed software projects.

The focus in SPM processes has thus clearly shifted from the position that it held two decades ago. Consequently, the size, complexity and strategic importance of information systems currently being developed require stringent measures to determine why projects might fail. Project or software quality management concerns itself with the prevention of failure and discrepancies. The purpose of quality management is to ensure that the product satisfies the needs of the stakeholders. As organizations continue to invest time and resources in strategically important software projects, software quality management becomes a critical area of concern.

Software agent technology offers a promising solution to addressing software quality management problems in a distributed environment. According to this technology, software agents are used to support the development of SPM systems in which data, control, expertise, or resources are distributed. Software agent technology provides a natural metaphor for support in a distributed team environment, where software agents can help the project manager and team members to monitor and coordinate tasks, to apply quality control measures, to validate and verify, as well as to ensure proper change control. SPM skills, especially in the distributed computing environment, are greatly in demand. Moreover, there is a need for technologies and systems to support the quality management of software projects in these environments. Our research is therefore aimed at software practitioners and software developers, but will also be beneficial to researchers working in the field of SPM.

In this paper the use of software agents is investigated as a potential tool for improving the quality management of SPM processes. We specifically concern ourselves with the question of how software agents can be used to improve quality management in a distributed environment. After investigating the various SPM key processes and some factors impacting on software quality management we propose a software agent framework to support software quality management. Although our research is not yet complete, initial indications are that it will enable software developers to meet market expectations and to manage risk factors accordingly. This, in turn, will bring about savings in cost, time and effort.

The next section contains a background study and a discussion on software quality management in the context of the software project management framework. The following section presents a topology of existing standards and measures for software quality management. We then provide

background information on agent technology, whereas the next section provides a generic-type multi-agent framework for quality management. This framework can be adapted to support all SPM processes and further extended using a (similar) multi-agent grid structure framework to coordinate the individual processes. We conclude by speculating that the proposed framework for enhancing software quality management may also be adapted to address other key SPM processes.

Software Project Management Background

Software Quality Management

The Project Management Body of Knowledge (PMBOK) defines project quality management as the processes required to ensure that the project will satisfy the needs for which it was undertaken. It includes all activities of the overall management function that determine the quality policy, objectives and responsibility, and implements these by means of quality planning, quality assurance, quality control and quality improvement, within the quality system. Quality management not only includes the concepts, tools and methods of quality assurance, but also validation and verification, as well as change control during the development process.

Major quality management processes identified by Schwalbe (2004) are:

- *Quality planning:* determining which quality standards are relevant to this specific project and deciding how these standards will be met.
- *Quality assurance*: involves evaluating overall performance regularly to ensure conformance to the set standards. Quality audits or reviews can support this function.
- *Quality control*: monitoring the activities and end results of the project to ensure compliance with the standards utilizing various available tools and techniques.

However, quality management should not be considered as a separate developmental phase but should be an inextricable part of all phases and all processes during software project management.

SPM is defined as the process of planning, organizing, staffing, monitoring, controlling and leading a software project (IEEE Standards Board, 1987). A more detailed exposition shows that SPM involves the planning, monitoring and controlling of people and processes that are involved in the creation of executable programs, related data and documentation (Elec 4704, 2003). Figure 1 illustrates a framework of the key elements in SPM identified by the PMBOK (Schwalbe, 2004). We distinguish between three key elements: project stakeholders, project management knowledge areas, and project management tools and techniques.

Project stakeholders comprise all the people involved in the different project activities and include the project sponsor, project team, support staff, customers, users, suppliers and even opponents. Good relationships, as well as communication and coordination between all of these stakeholders, are essential to ensure that the needs and expectations of stakeholders are understood and met. Knowledge areas include the key competencies involved in the software project management process. The core functions, namely scope, time, cost and quality management lead to specific project objectives and are supported by the facilitating functions. The facilitating functions represent the means through which different objectives are to be met and include human resource management, communication, risk and procurement management. Reaching across all these knowledge areas are the project management tool and techniques (see Figure 1). These are used to assist team members and project managers in carrying out their respective tasks.

However, traditional tools and techniques supporting the key management areas are not adequate (Chen, Nunamaker, Romano, & Briggs, 2003) in a coordinated, distributed team-development environment. The 1995 Standish Group study found that the three major factors related to information technology project success were user involvement, executive management support and a clear statement of requirements (Standish Group, 1995). In the 2000 report of the Standish Group, executive management support, user involvement, an experienced project manager and a clear statement of requirements top the list of requirements for success. Software quality management can address and improve all of these aspects.

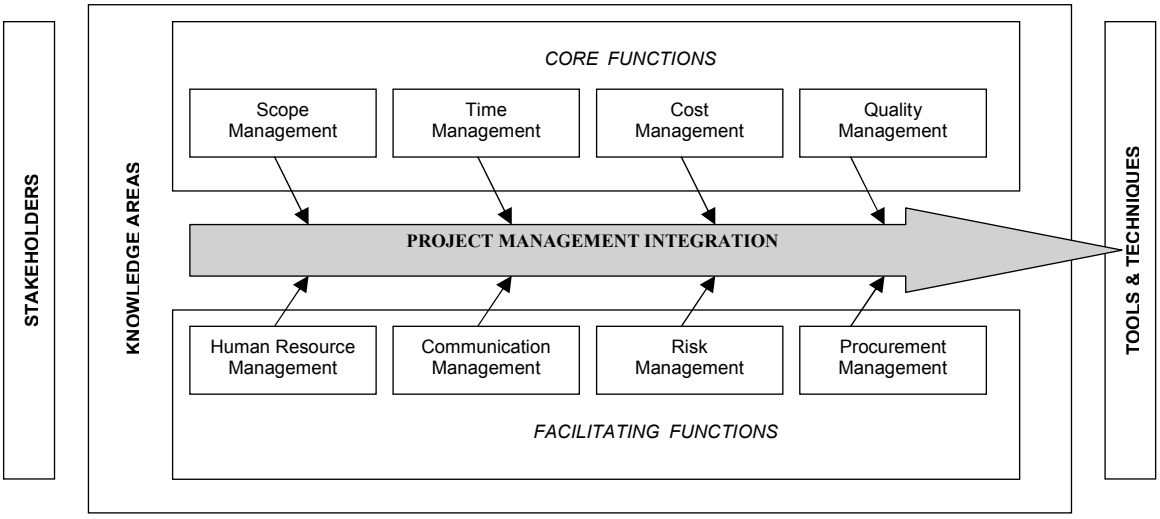


Figure 1: *Software Project Management Framework (adapted from Schwalbe, 2004).*

Hughes and Cotterel (2002) recommend that quality aspects of the project plan should be reviewed constantly. In considering the three phases of quality management, the following phases are of importance: *The quality-planning phase* should identify variables having a direct influence on the outcome of the project. Thus, aspects affecting the scope of the project are functionality, system outputs, performance and reliability. All these factors should be included in the quality assurance plan. *The quality assurance phase* involves evaluation measures throughout the project. Tools used include quality audits, templates specifying required documentation, quality assurance procedures, problem-reporting procedures, quality assurance metrics and quality assurance check list forms. *The quality control phase* mainly consists of the following: acceptance decisions to determine whether the products or services produced will be accepted or rejected; if not accepted, rework specified on the items; and process adjustments to correct or prevent further quality problems. Various tools and techniques may be utilized during this phase. Process as well as product quality measures should be implemented. Several standards and measures have been developed over the past few years in an effort to give structure and uniformity to this process. These standards will be discussed in the following section.

Existing Standards and Measures

Various factors enhancing quality have been identified over the years in an attempt to improve quality measures, but lack of conformity of definitions and terms posed a problem. Software development is a fast growing industry and the lack of standards has significant implications for society and the economy. In an attempt to solve this problem various national and international standards bodies proceeded to set standards for this area of development.

The PMI (Project Management Institute) coded the Project Management Body of Knowledge's (PMBOK) first published standard in 1983, namely the Project Management Quarterly Special Report: Ethics, Standards and Accreditation. This was further developed and the PMBOK Standards were published in 1987, whilst the Guide to Project Management Body of Knowledge was published in 1996. Currently PMI is working on the OPM3 standard, as the global standard for organizational project management.

The ISO standard 9126 was published in 1991 (Hughes & Cotterel, 2002) to address the problem of defining software quality. ISO 9126 identified six software quality characteristics, namely functionality, reliability, usability, efficiency, maintainability and portability. Sub-characteristics for each of these are also identified. Measurements correlating to each quality are identified, and then tested and mapped onto a scale to indicate compliance with the specific quality metric.

The British Standards Institution set the BS EN ISO 9001:2000 standard, identical to the international standard ISO 9000:2000, followed by the 2001 and 2004 standards respectively.

The capability maturity model (CMM) was developed at the Software Engineering Institute in the United States. This model defines different stages of process maturity, implying sophistication and quality of production practices in which an organization may find itself. The assessment is done by an external team of assessors, who will also make recommendations on improving the quality processes. Bootstrap, a European initiative, allows assessment at project level.

Hughes and Cotterel (2002) define practical software quality measures, such as reliability, which might measure availability, mean time between failures, failure on demand and support activities. Other practical measures are maintainability and extendibility.

Measurement of quality concerns intangible, invisible factors. Techniques to enhance quality (Hughes & Cotterel, 2002) are cited as increased visibility, procedural structure and checking of intermediate stages:

Increasing visibility of the development process consists of utilizing ego-less programming to encourage the practice of programmers scanning each other's code.

Procedural structure implies the use of methodologies, where every process in the development cycle has carefully laid out plans.

Checking intermediate stages involves the continuous checking of quality and correctness of work done throughout the development phases.

Other techniques recommended are inspections, structured programming and clean-room software development, formal methods and software quality circles.

Different approaches to quality control are also utilized. Mehandjiev et al. (2002) state that a goal-driven approach is more appropriate to handle adaptability and productivity requirements, whereas Szejko (2002) promotes requirements-driven quality control.

Using Agent Technology to Enhance Quality Standards

Agent Technology

A software agent is a computer program that is capable of autonomous (or at least semi-autonomous) actions in pursuit of a specific goal (Krupansky, 2003). The autonomy characteristic of a software agent distinguishes it from general software programs. Autonomy in agents implies that the software agent has the ability to perform its tasks without direct control, or at least with minimum supervision, in which case it will be a semi-autonomous software agent. Software

agents can be grouped, according to specific characteristics, into different software agent classes (d'Inverno & Luck, 2001). Literature does not agree on the different types or classes of software agents. As software agents are commonly classified according to a set of characteristics, different classes of software agents often overlap, implying that a software agent might belong to more than one class at a time. For the purpose of this research, we distinguish between two simple classes of software agents, namely stationary agents and mobile agents. Agents in both these classes might, or might not have, any or a combination of the following characteristics: a user interface, intelligence, adaptivity, flexibility and collaborative properties (Pacheco & Carmo, 2003).

Whether or not an agent has a user interface, depends on whether it collaborates with humans, other agents or hosts. User interfaces are commonly only found where agents interact with humans. According to Wooldridge (2001), intelligence implies the inclusion of at least three distinct properties, namely reactivity, pro-activeness and social ability. *Reactivity* refers to the agent's ability to perceive its environment and respond timeously to changes that occur in order to achieve its design goals. *Pro-activeness* is the agent's ability to take the initiative in its environment in order to achieve its design goals. *Social ability* alludes to the collaborative nature of the agent. There are different definitions to define the collaborative nature of software agents. For the purpose of this paper we use Croft's (1997) definition in which the collaborative nature of a software agent refers to the agent's ability to share information or barter for specialized services to cause a deliberate synergism amongst agents. It is expected of most agents to have a strong collaborative nature without necessarily implying other intelligence properties. *Adaptivity* is a characteristic that can also be regarded as an intelligence property, although it is not counted as a prerequisite for identifying an agent as intelligent. Adaptivity refers to an agent's ability to customize itself on the basis of previous experiences. An agent is considered flexible when it can dynamically choose which actions to invoke, and in what sequence, in response to the state of its external environment (Pai et al. 2000).

A stationary agent can be seen as a piece of autonomous (or semi-autonomous) software that permanently resides on a particular host. An example of such an agent is one that performs tasks on its host machine such as accepting mobile agents, allocating resources, performing specific computing tasks, enforcing security policies and so forth. A well known example of a stationary agent is Clippie, the Microsoft Office Assistant. Clippie exhibits similar features of a stationary, intelligent agent and its settings are global for all programs in the Microsoft Office Suite.

A mobile agent is a software agent that has the ability to transport itself from one host to another in a network. The ability to travel allows a mobile agent to move to a host that contains an object with which the agent wants to interact, and then to take advantage of the computing resources of the object's host in order to interact with that object. An example of a mobile agent is provided by a flight booking system where a logged request is transferred to a mobile agent that on its part traverses the web seeking suitable flight information quotations as well as itineraries. Full autonomy, migratability and collaborativeness are the most important characteristics that should be embedded in each mobile agent. When a mobile agent possesses these three intelligence requirements, it is often referred to as a 'robot' (Krupansky, 2003). For a more detailed discussion of mobile agent systems and associated design concepts refer to Schoeman and Cloete, 2004.

Software Agents in SPM

Software agent technology is being explored as a promising way to support and implement complex distributed systems. In this section, the authors briefly consider how agent technology is currently being deployed in SPM by considering some application examples. As described earlier, the software project management environment has changed in the past decade into a dynamic and

complex environment in which flexible and adaptive behaviour and management techniques are required. Agent-based solutions are most applicable to this environment since they are appropriate in highly dynamic, complex, centralised as well as distributed situations. In addition to the advantages of distributed and concurrent problem solving, agent technology has the advantage of sophisticated patterns of interaction, namely cooperation, coordination and negotiation (Hall, Guo, & Davis, 2003).

The first application that we mention utilizes agents for project planning and process management in a distributed environment. O'Connor and Jenkins (1999) propose an intelligent assistant system to assist the project team during planning, scheduling and risk management.

In a second example software agents are used to control and monitor activity execution at various sites in an open source platform supporting distributed software engineering processes. This environment is being developed as part of the GENESIS project (Gaeta & Ritrovato, 2002). Although this project does not relate to quality management, it uses agents to support the control of software processes as well as the communication among distributed software engineering teams. Agents are mainly utilized for the synchronizing of process instances executed on different sites, the dynamic reconfiguration of software processes, process data collection, monitoring of the processes and artefact retrieval. Other relevant examples of agent utilization in SPM can be found in Maurer (1996) and Sauer and Appelrath (2003). Sauer and Applerath (2003) presented an application using agents to focus primarily focus on the Time Management function and certain aspects of the Communication Management function. Maurer's solution (1996) is applicable to the Scope Management, Time Management and to a certain extent the Communication Management functions.

Multi-Agent Model for Software Quality Management

We briefly reconsider the distinct knowledge areas and practices that in software project management entails (illustrated in Figure 1), to emphasise the focus of our work for this paper. The SPM management areas consist of four objective functions and four facilitator functions. We believe that each of these key processes/functions could successfully be addressed by following a black-box approach that is based on agent technology. Each black box consists of collaborative software agents ensuring cooperation, coordination and synergy between the different black boxes. Within such a black box a component-based development approach is followed. According to this approach, we use multiple (simple) agents (discussed on the following page), each with a particular objective, rather than fewer (complex) agents of which each has a long list of tasks to accomplish. For the purpose of this paper, we discuss our approach to only one of the SPM key processes, namely Quality Management, and describe the agent framework to accomplish the black-box for this process. (Models for the communication management and risk management function can be consulted in previous work of the authors.)

To describe how software agents are used to address the different functions of *quality management*, we use a set of *agent teams* to address the individual functions and then define specialised software agents operating within these teams, or on their own where applicable. In defining these specialised software agents, we find that it is less intricate to design the behaviour of each agent. Furthermore, the specialised agents also make it possible to describe the various interactions between different agents explicitly, which in turn reduces the general complexity of the agent system. The various programming patterns (Aridor & Lange 1998; Kendall, Krishna, Suresh, & Pathak, 2000) available, accomplish specific agent-associated tasks, such as creation, migration, suspension, and collaboration.

The design of the overall system, based on components (specialised agents), simplifies the design and programming of agents. The following specialised working agents are used in our discussion

of the quality management model that we present in the next subsection. These working agents include:

Personal assistant agent (PA agent): an agent that supports an individual stakeholder to accomplish his or her tasks by providing maximum assistance. This agent also has a collaborative nature, and relies on other agents to provide it with the information that it requires to sustain its owner. The PA agent is not computer-bound, but human-bound, as its human stakeholder may work on different computers in a distributed environment.

Messaging agent: is an agent responsible for carrying messages between different agent teams. A messaging agent has strong collaborative characteristics and is by nature a mobile agent since the different agent teams may function in a distributed environment.

Task agent: an agent that supports a specific project task. This agent collaborates with other objective and facilitator functions to support a specific task. This agent is commonly invoked by a PA agent to allow a stakeholder to work on a specific task, and is continuously monitored by a monitoring agent.

Monitoring agent: is an agent responsible for monitoring tasks. A monitoring agent is mobile, with intelligence, flexibility and strong collaborative properties.

Team manager agent: an agent that is responsible for managing a team of agents, ensuring coordination between the subtasks of the different members of a team to accomplish the objective of the agent team.

Figure 2 illustrates the main operations in the quality management function. Various agents, as described above will be developed and utilized to support every function of quality management. Agent teams will cooperate to accomplish the objectives of these functions.

The interaction between different functions are depicted by arrows illustrating the direction of the interaction.

For the model we present in this paper, we will adopt a combination of these functions:

Quality planning consists of determining which quality standards are relevant to this specific project and deciding how these standards will be met. Obviously a quality plan must be devised and set. In our discussion we assume quality measures derived from: (1) requirements and (2) standards. Agents utilized will be:

Task agents to set and identify relevant quality measures,

mobile agents to communicate to stakeholders and teams,

monitoring agents to receive and distribute

teamwork agents to coordinate agents.

Quality assurance involves evaluating overall performance regularly to ensure conformance to the set standards. Quality audits or reviews can support this function. Agents involved will be:

Task agents to evaluate compliance to set relevant standards, and give warning messages,

mobile agents to communicate,

messaging agents to deliver messages,

monitoring agents to control and execute audits.

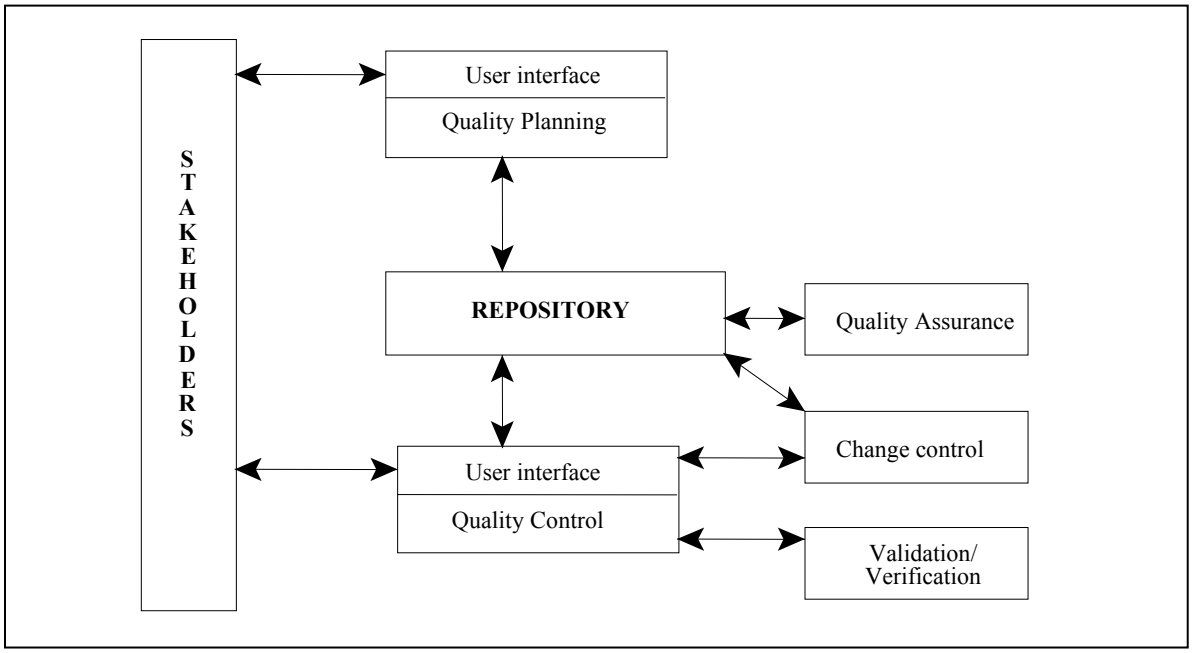


Figure 2: *Software Project Quality Management Framework.*

Quality control involves monitoring the activities and end results of the project to ensure compliance with the standards utilizing various available tools and techniques.

Task agents to execute monitoring tasks,

mobile agents to receive and carry information,

messaging agents, personal agents,

monitoring agents to control and check that tasks meet measures.

Conclusion

In this paper we investigated an approach of using software agent technology to address the challenges posed in the software project management arena. We focused on one of the key elements of SPM, namely software quality management, and designed a generic agent framework to address all the tasks of this key element. This framework forms a basis for other key elements, and could be adapted into individual frameworks and then coordinated by an overall multi-agent system to achieve the objectives of SPM. Our framework follows an approach of agent teams being composed of specialised software agents, each tasked with a manageable /atomic/ task. This implied that the complexity of creating and maintaining tasks could be greatly reduced. Although we have not yet completed the programming of the proposed system, we believe that our solution in the form of a framework can potentially be significant, based on our experience in other fields that advocate component-based development. We do, however, recognize the fact that programming of the model will have to be completed and the model thoroughly tested against other SPM tools before its true value will become apparent.

Acknowledgements

This research is based upon work supported by the National Research Foundation of South Africa under Grant Number (GUN 2054319) in cooperation with the University of South Africa (UNISA). Any opinion, findings and conclusions or recommendations expressed in this material are those of the authors and therefore the NRF does not accept any liability in regard thereto.

References

Aridor, Y., & Lange, D.B. (1998). Agent design patterns: Elements of agent application design. *Proceedings of the 2nd International Conference on Autonomous Agents*. Minneapolis/St. Paul, USA. 108 - 115.

Chen, F, Nunamaker, J. F., Romano, N. C. & Briggs, R. O. (2003). A collaborative project management architecture. *Proceedings of the 36th Hawaii International Conference on System Sciences.* Big Island, Hawaii.

Croft, D.W. (1997). Intelligent software agents: Definitions and Applications. Retrieved from http://www.alumni.caltech.edu/~croft/research/agent/definition/

D'Inverno, M. & Luck, M. (2001). *Understanding Agent Systems.* Berlin: Springer-Verlag.

Durham J. T. & Torrez, W. C. (2004). Net-Centric human-robotics operations. *Proceedings of the IEEE International Conference on Web Services (ICWS'04).* IEEE 2004.

ELEC 4704 - Software project management. Department of Electrical and Information Engineering. (2003). University of Sydney. Retrieved May 2004 from http://www.ee.usyd.edu.au/elec4704/lec-01.html

Hall, G., Guo, Y. & Davis, R. A. (2003). Developing a value-based decision-making model for inquiring organizations. *Proceedings of the 36th Hawaii International Conference on System Sciences.* Big Island, Hawaii.

Hass, A. M. J., Johansen, J., & Pries-Heje, J. (1998). Does ISO 9001 increase software development maturity. *Proceedings of the 24th EUROMICRO Conference.* 1089-6503/98 1998 IEEE.

Hughes, B. & Cotterel, M. (2002). *Software project management* (3rd ed.). McGraw-Hill.

IEEE Standards Board. (1987). *IEEE Standard for software project management* Plans. IEEE Std 1058.1-1987. 16pp. PDF: ISBN 0-7381-0409-4, SS12138.

Kendall, E. A., Krishna, P. V., Suresh, C. B. & Pathak, C. V. (2000). An application framework for intelligent and mobile agents. *ACM Computing Surveys, 32*, 1.

Krupansky, J. W. (2003). *What is a Software Agent.* Website. Retrieved May 2003 from http://agtivity.com/agdef.htm

Marchewka, J.T. (2003). *Information technology project management.* Wiley.

Maurer, F. (1996). Project coordination in design processes. *Proceedings of the 5th International Workshops for Enabling Technologies*: Infrastructure for Collaborative Enterprises (WET ICE'96) IEEE.

Mehandjiev, N., Layzell, P., Brereton, P., Lewis, G., Mannion, M, & Coallier, F. (2002). Thirteen knights and the seven-headed dragon; An interdisciplinary software engineering framework. *Proceedings of the 10TH International Workshop on Software Technology and Engineering Practice (STEP '02).* 2002 IEEE.

O'Connor, R. & Jenkins, J. (1999). Using agents for distributed software project management. *Proceedings of 8th International Workshop on Enabling Technologies*, pp 54-60, IEEE Computer Society Press.

Pacheco, O. & Carmo, J. (2003). A role based model for normative specification of organized collective agency and agents interaction. *Journal of Autonomous Agents and Multi-Agent Systems, 6* (2), 145-184. Kluwer Academic Publishers.

Pai, W. C., Wang, C. C., & Jiang, D. R. (2000). A software development model based on quality measurement. *Proceedings of the ICSA 13th International Conference*. Computer Applications in Industry and Engineering, 40-43.

Sauer, J, & Applerath, H. (2003). Scheduling the supply chain by teams of agents. *Proceedings of the 36th Hawaii International Conference on System Sciences.* Big Island, Hawaii.

Schoeman, M. A, & Cloete, E. (2004). Design concepts for mobile agents. *SA Computer Journal, 32.*

Schwalbe, K. (2004). *Information technology project management* (3rd ed.). Thompson Learning.

Szejko, S. (2002) Requirements driven quality control. *Proceedings of the 26TH International Computer Software and Applications Conference* (COMPSA).

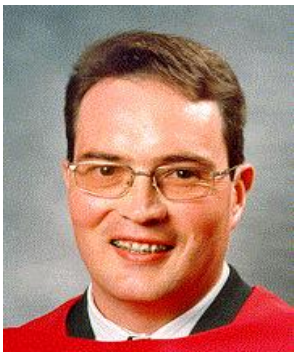
The Standish Group. (1995).CHAOS. Retrieved 2003 from http://www.standishgroup.com/

The Standish Group. (2000). EXTREMECHAOS. Retrieved April 2003 from http://www.standishgroup.com/

Wooldridge M. (2001). *An introduction to multi-agent systems*. Chichester, UK: John Wiley & Sons.

Biographies

Ms **Rita C Nienaber** is currently a senior lecturer at the University of South Africa in the Department of Software Development, School of Computing. She obtained her Master of Science (Information Technology), from the University of South Africa in 1996 and is currently enrolled for a doctorate degree at Unisa. Whilst lecturing on modules namely database systems development, system analysis and design and software project management, her areas of publishing focus on software project management and software agent technology.



Andries Barnard, associate professor in the Department of Computer Science and Information Systems, University of South Africa, holds a PhD (Computer Science). He teaches undergraduate courses in automata theory and formal languages and project management, as well as postgraduate courses in project management and research methodology. His research interests include software project management and software agent technology, computer ethics as well as graph grammar languages.