Interdisciplinary Journal of Information, Knowledge, and Management Volume 2, 2007

A Generic Agent Framework to Support the Various Software Project Management Processes

Rita C Nienaber and Andries Barnard
School of Computing, University of South Africa, Pretoria, South Africa

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

Abstract

Despite various research efforts originating from both academia and industry, software projects have a high rate of failure, more specific, software projects often do not comply with the traditional standard measurements of success, namely time, cost and requirements specification. Thus, there is a need for new methods and measures to support the software project management process.

Globalisation and advances in computing technologies has changed the software project management environment. Currently software projects are developed and deployed in distributed, pervasive and collaborative environments and traditional project management methods cannot, and do not, address the added complexities inherent to this environment.

In this paper the utilisation of stationary and mobile software agents is investigated as a potential tool to assist with the improvement of software project management processes. In particular we propose and discuss a software agent framework to support software project management. Although still in its initial phases, this research shows promise of significant results in enabling software developers to meet market expectations, and produce projects on time, within budget and to users' satisfaction.

Keywords: Software Project Management, Software Agent Technology, Project Scope Management, Project Time management, Project Cost Management, Project Quality Management, Project Risk Management, Project Communication Management, Project Human Resource Management, Project Procurement Management,

Introduction

Software Project Management (SPM) has become a critical task in many organisations. Managing software projects is a complex task, further complicated by a continued increase in the size and complexity of the software-intensive system. In the 1980's SPM methodologies primarily focused on providing schedule and resource data to management (Schwalbe, 2006.) However, present-day SPM activities involve much more. With the advent of the Internet, improvement of computer hardware, software, and networks, global interdisciplinary work teams have changed the working environment addressed by SPM. Global networking capabilities have become more pervasive with the result that cost-

Editor: Kathy Lynch

effective computing resources will continue to play a major role in improving organisational operations.

SPM involves the management of all issues involved in the development of a software project, namely scope and objective identification, evaluation, planning, project development approaches, software effort and cost estimation, activity planning, monitoring and control, risk management, resource allocation and control, as well as managing contracts, teams of people and quality.

Since publication of the 1995 report of The Standish Group, this same organisation studied 13,522 projects in a follow-up survey, aptly dubbed EXTREME CHAOS (The Standish Group, 2000). 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 reports that success rates increased to a third of all projects, but time overruns increased to the 82nd percentile, whilst only 52 percent of required and specified functions and features were included in the final product (The Standish Group, 2003).

Many software projects still failed to comply with the triple constraints of scope, time and cost (Oghma: Open Source, 2003). These triple constraints refer to the fact that the failure of software projects can mostly be attributed to projects not delivered on *time* and that it does not meet the expectations of the client *(scope)*, and as a result have *cost* overrun implications. As previously mentioned, the SPM environment is continuously changing due to globalisation and advances in computing technology. This implies that the traditional single project, commonly executed at a single location, has evolved into distributed, collaborative projects. The focus in SPM processes has 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 ensure that software projects do not fail. As organisations continue to invest time and resources in strategically important software projects, managing the risk associated with the project becomes a critical area of concern.

Software agent technology offers a promising solution in order to address SPM 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 support the project manager and team members to monitor and coordinate tasks, apply quality control measures, validation and verification, as well as change control. Agent technology has distinct advantages over client/server technology as distributed system instantiation. SPM skills, especially in the distributed computing environment, are greatly in demand. Moreover, there is a need for technologies and systems to support management of related aspects of software projects in such 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 to improve the management of related SPM processes. We specifically concern ourselves with the question of how software agents can be used to improve all core and facilitating management functions in distributed environments. As a result, we propose two software agent frameworks to support SPM in such environments. Although our research is not yet complete, initial indications are that it will enable software developers to meet market expectations and to manage risk and associated core and facilitating factors accordingly. This, in turn, will bring about savings in cost, time and effort.

In the next section of this paper, brief information regarding agent technology is provided. The third section contains a background study on SPM and a discussion on agents utilised in SPM. In the fourth section the phases of the core and facilitating functions during SPM are discussed, as

well as a proposal of a generic multi-agent framework supporting SPM. This framework supports the entire spectrum of SPM processes and as instantiation thereof, has been conformed to include our previously identified frameworks for risk, quality and communication management (Nienaber & Barnard, 2005; Nienaber & Cloete, 2003; Nienaber, Cloete, & Barnard, 2004). Finally, a conclusion is presented.

Software Agent Technology

This section presents a discussion on software agent technology. Differentiating properties of software agents are explained.

A software agent is a software program that is capable of autonomous (or at least semi-autonomous) actions in pursuit of a specific goal. 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. 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 (d'Inverno & Luck, 2001). 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 may, or may not have, any or a combination of the following characteristics: a user interface, intelligence, adaptivity, flexibility and collaborative properties (Wooldridge, 2001).

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 software agents are required to interact with humans. According to Wooldridge (2001) intelligence implies the inclusion of at least three distinct properties, namely reactivity, proactiveness and social ability. *Reactivity* refers to the agent's ability to perceive its environment and respond in a timely manner to changes that occur in order to achieve its design goals. *Proactiveness* is the agent's ability to take 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 specialised 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 to identify 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, Wang & Jiang, 2000).

A stationary agent can be seen as a piece of autonomous (or semi-autonomous) software that permanently resides on a particular host. Such an agent performs tasks on its host machine such as accepting mobile agents, allocating resources, performing specific computing tasks, enforcing security policies and so forth.

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 itself 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. Full autonomy, migratability and collaborativeness are the most important characteristics that should be imbedded in each mobile

agent. When a mobile agent possesses these three intelligence requirements, it is often referred to as a robot (Krupansky, 2003).

Software Project Management (SPM)

Software Project Management Framework

SPM is defined as the process of planning, organising, 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 Project Management Body of Knowledge (PMBOK),* (Project Management Institute, 2004). We distinguish between three key elements: project stakeholders, project management knowledge areas, and project management tools and techniques.

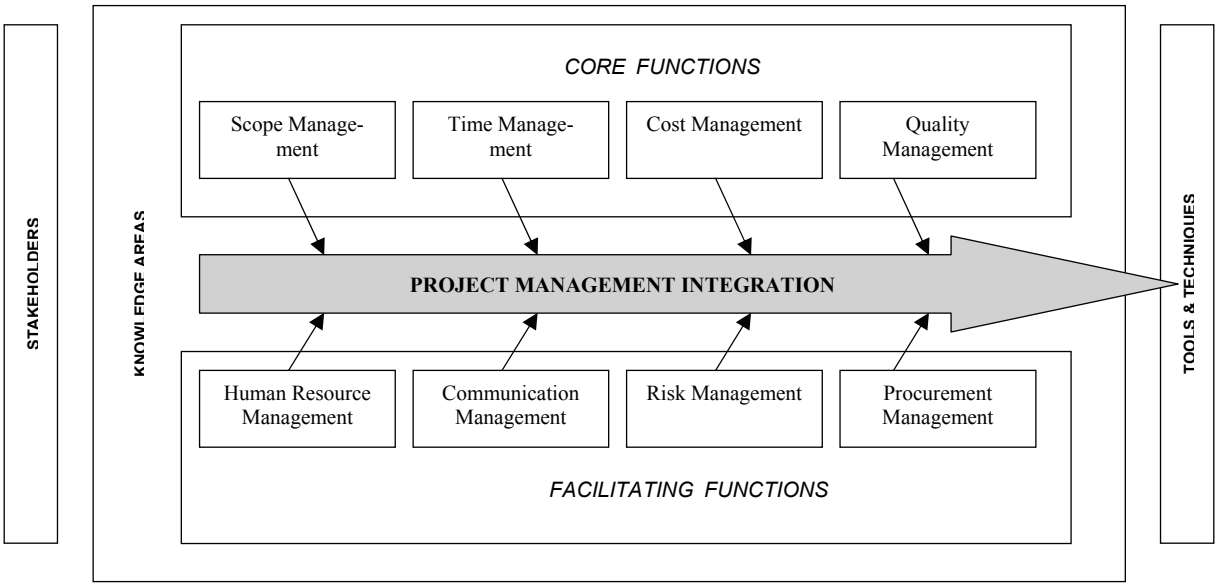


Figure 1: Software Project Management Framework (adapted from Schwalbe, 2006)

Project stakeholders are those individuals involved in all different project activities and include the project sponsor, project team, support staff, customers, users, suppliers and even opponents of the project. Although these stakeholders may have different views and expectations, 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.

Software project management knowledge areas include the key competencies concerned during the software project management process. These areas are categorised as core and facilitating functions. 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. Stretched across all these knowledge areas are the project management tools and techniques (on the right-hand side of the framework diagram). These are used to assist team members and project managers in carrying out the core and facilitating functions.

Software Agents in SPM

Software agent technology is at present explored as a promising way to support and implement complex distributed systems and a useful supplement to client/server systems. In this section, the authors briefly consider how agent technology is currently deployed in SPM by considering some application examples. As described earlier, the SPM environment has changed in the past decade into a dynamic and complex environment where flexible and adaptive behaviour and management techniques are required. Agent-based solutions are applicable to this environment since they are appropriate in highly dynamic, complex, centralised as well as distributed situations (Dowling, 2000). 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 utilises agents for project planning and process management in a distributed environment. O'Connor & Jenkins (1999) propose an intelligent assistant system to support the project team during planning, scheduling and risk management. Joslin & Poole (2005) adapts a simulation-based planning algorithm to the problem of planning for SPM.

In another 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). Software agents are used in this project to support the control of software processes as well as the communication among distributed software engineering teams. Agents are mainly utilised for the synchronisation of process instances executed on different sites, the dynamic reconfiguration of software processes, process data collection, monitoring of the processes, as well as artefact retrieval. Other relevant examples of agent utilisation in SPM can be found, among others, in Maurer (1996) and Sauer & Appelrath (2003). Sauer & Applerath (2003) present an application using agents to primarily focus on Time Management and certain aspects of the Communication Management function. Maurer's solution (1996) is applicable to Scope Management, Time Management and, to a certain extent, the Communication Management function. Agent technology has been more commonly applied to areas such as network and system management (Kendall, Krishna, Suresh & Pathak, 2000), decision and logic support (Burstein, McDermott, & Smith, 2000), interest matching (Object Management Group, 2000), data collection in distributed and heterogenous environments, searching and filtering, negotiating, and monitoring (Venners, 1997).

Multi-Agent Model for Software Project Management

Software Project Management Phases

In order to identify and compile a general multi-agent model to facilitate (in the following two sections) all of the SPM processes involved, the steps comprising each process of each of the key areas will be elaborated on below:

Software scope management:

Schwalbe (2006) identifies the following specific phases of software project scope management namely initiation, scope planning, scope definition, scope verification and scope change control. *Initiation* of the project involves the commitment of an organisation to a project. *Scope planning* identifies and refines project scope and creates a formal scope statement document, s*cope definition* involves the division of major project deliverables into smaller and more manageable components and s*cope verification* includes formal acceptance of the scope of the project by the various key stakeholders.

Software time management:

Time management involves the processes required to measure timely completion of a project and as such involves not only the creation of an activity plan, but also the estimation of each task and activity, resulting in the overall duration of the project. *Activity planning* constitutes the baseline for project and resource scheduling, supporting a number of objectives (Hughes & Cotterel, 2006), namely feasibility assessment, resource allocation, detailed costing, motivation and coordination of the project. The main processes involved in time management (Schwalbe, 2006) are briefly reflected on below:

Activity definition involves the identification of each task or activity that must be executed in order to produce the project deliverables. *Activity sequencing* indicates when each of the identified activities should occur. *Activity duration estimation* concerns estimating the work periods to be executed. *Schedule development* involves utilising the previous two activities, as well as resource requirements, to create the project schedule. *Schedule control* refers to the controlling and managing of changes to the initial schedule.

Software cost management:

Cost management can be seen as all processes required to ensure that a project team completes a project within an approved budget (Schwalbe, 2006). *Cost estimation* refers to the process of developing an approximation or estimate of the costs of all actions, resources and procedures, and *cost budgeting* involve using the project cost estimate and allocating this to individual work items. *Cost control* and *monitoring* includes monitoring cost performance, reviewing changes and notifying stakeholders and team members of changes related to cost.

Software quality management:

The Project Management Body of Knowledge (PMBOK) defines project quality management as 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. Major quality management processes identified by Schwalbe (2006) are q*uality planning* during which quality standards are identified and applied. *Quality assurance* involves evaluating overall performance regularly, quality audits or reviews can support this function. *Quality control* concerns monitoring activities and end results to ensure compliance to standards.

Software human resource management:

Human resource management involves all processes required to effectively utilise all resources involved in a project. A resource may be seen as any item or person required for the execution of a project. Human resource management concerns all project stakeholders involved in developing the project. The main focus of this process is to allocate resources to activities, and to create a work schedule from the activity plan. Hughes & Cotterell (2006) identifies seven categories of resources to be managed for a project, namely: labour, equipment, materials, space, services, time and money. Schwalbe (2006) identifies three phases, namely organisational planning, staff acquisition and team development.

Software communication management:

Communications management in a software project is an enabling and supporting action that ensures timely and appropriate generation, collection, dissemination, storage and disposition of project information (Schwalbe, 2006). Effective communication and sharing of information and knowledge among project contributors are required. Schwalbe identifies five distinct functions

associated with communications management, namely: The *communications planning* function that determines the *who*, *when* and *how* of the project, whilst the *information distribution* function entails disseminating information to keep all stakeholders informed. *Performance reporting* alludes to the generation of reports such as status, progress and forecasting reports, while the *administrative closure* function involves project archiving and formal acceptance of reports. Finally the *teamwork support* function refers to the functions pertaining to collaborative project tasks, and hence includes the scheduling of meetings for these collaborative tasks. It therefore facilitates a collaborative working environment as well as document distribution.

Software risk management:

Various models or frameworks exist to ameliorate the risk associated with software project development. According to Marchekwa (2003), this basically entails two aspects, namely risk analysis and risk management. *Risk analysis* includes risk identification, qualitative and quantitative risk analysis, evaluation and assessment. *Risk management* on the other hand entails risk planning, monitoring and control. Similarly, Hughes and Cotterel (2006) identify two major areas, namely risk analysis and risk management, based on Boehm's model (1989), including the following functions namely risk identification, risk evaluation, risk planning, risk control, and risk monitoring

Software procurement management:

During the process of software project development, products, goods or items that are not readily available within the organisation (perhaps in the form of software, hardware or people) must be acquired (Marchewka, 2003). Procurement refers to the process of acquiring goods or services from an outside source. Procurement management thus entails a set of procedures to facilitate acquisition of such products, expediting external work and to ensure the satisfactory standard of work throughout a given organisation. These may involve rules for acquisition, purchase order documentation required by a specific organisation and creating and maintaining lists of trustworthy, qualified vendors (Hughes & Cotterel, 2006). Project procurement management consists of six main processes, namely procurement planning, solicitation planning, solicitation, source selection, contract administration, and contract close-out (Schwalbe, 2006).

However, these phases should not be considered as separate development phases but should be entwined in all phases and all processes during the SPM undertaking. Table 1 depicts the phases utilised during execution of the core and facilitating functions:

Table 1: Software Project Management core and facilitating functions

Scope Management	Time Management	Cost Management	Quality Management	HR Management	Communication Management	Risk Management	Procurement Management
Initiation	Activity definition	Cost & resource planning			Identification	Identification	Identification
Definition	Activity sequencing Activity duration estimation	Cost estimation	Planning	Planning Team development	Planning Team support	Estimation Evaluation Assessment	Solicitation Planning
Planning	Time schedule development	Budgeting	Assurance	Monitor & control	Information Distribution	Planning Staffing	Contract administration
Control	Time schedule control	Control	Control		Performance Reporting	Monitor Control	Control
Verification			Validation		Admin closure		

As abstraction of this table the correlating phases of the core and facilitating functions will be used to compile a generic model in a subsequent section.

Software Agents to Support SPM

Software agent technology provides a useful paradigm for the use of distributed computational resources. Mobile agents (Butte, 2002) enable a shift in the communication paradigm of distributed systems from data shipping to function shipping. Using mobile agent technology, in comparison to the classic well-known Remote Procedure Call (RPC), or its object-oriented equivalent Remote Method Invocation (RMI), due to the autonomous code it entails may attain a higher level of abstraction. This autonomy reduces network load and communication overhead in distributed applications. Distributed applications based on RPC techniques are suitable for stable and static system structures, which is not always the case in a distributed environment.

To describe how software agents are used to address the different functions, 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 et al., 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 generic multi-agent framework 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: an agent responsible for transporting 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. Such an 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: 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 sub-tasks of the different members of a team to accomplish the objective of the agent team.

For the model we present in this paper, we will adopt a combination of these agents.

Software Agent Framework to Support SPM

We briefly reconsider the distinct knowledge areas and practices entailed in software project management (illustrated in Figure 1 and summarised in Table 1), to emphasise the focus of our

work for this paper. The SPM areas consist of four core 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, each with a particular objective, rather than fewer (complex) agents of which each has a long list of tasks to accomplish. An abstraction of the generic functions of a key SPM process was compiled into a generic model (Figure 2).

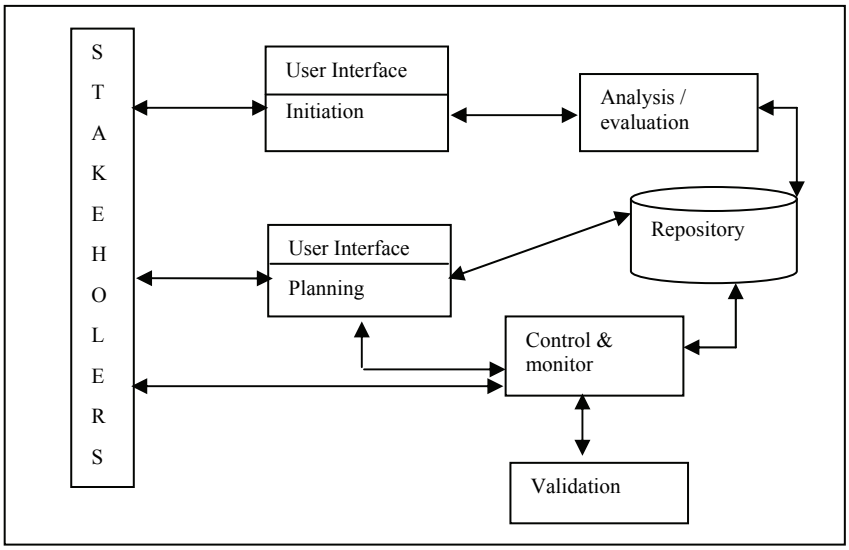


Figure 2: Generic model for SPM processes

This abstraction will be used to compile two generic multi-agent frameworks supporting all phases of SPM. In particular we discuss our approach to the entire spectrum of the SPM key processes, and describe the agent framework to accomplish the black-box for these processes.

As mentioned previously, an abstraction of the functions of the key SPM processes was compiled into a generic model (Figure 2). This abstraction can then be used to compile a conceptual model or framework for each of the key SPM processes. To illustrate this process risk management is used as an example. Software risk management consist of the following phases: risk identification, risk analysis that includes risk assessment and evaluation, risk planning, monitoring and control. An agent framework depicting this key area is illustrated in Figure 3.

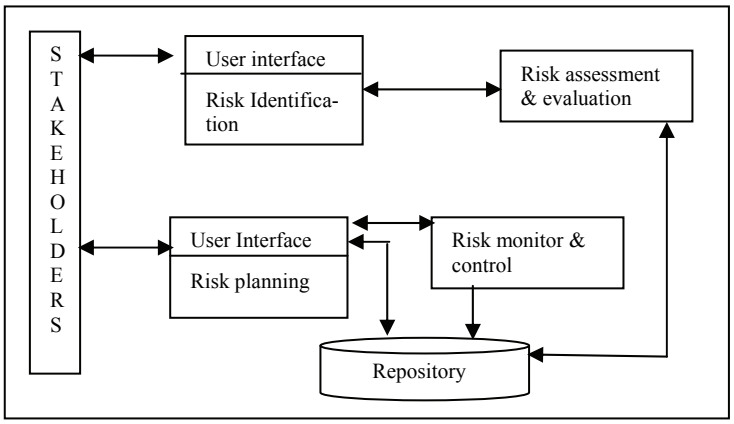


Figure 3: Software risk management

The generic model as depicted in Figure 2 was instantiated to one key area, namely risk management, resulting in Figure 3. In a similar way the basic generic model will be detailed, elaborated and expanded on to compile an overall framework and two conceptual models will be created depicting the core functions and the facilitating functions, Figure 4 and Figure 5 respectively. A conceptual model for the SPM core functions: time management, cost management, quality management & scope management is shown in Figure 4.

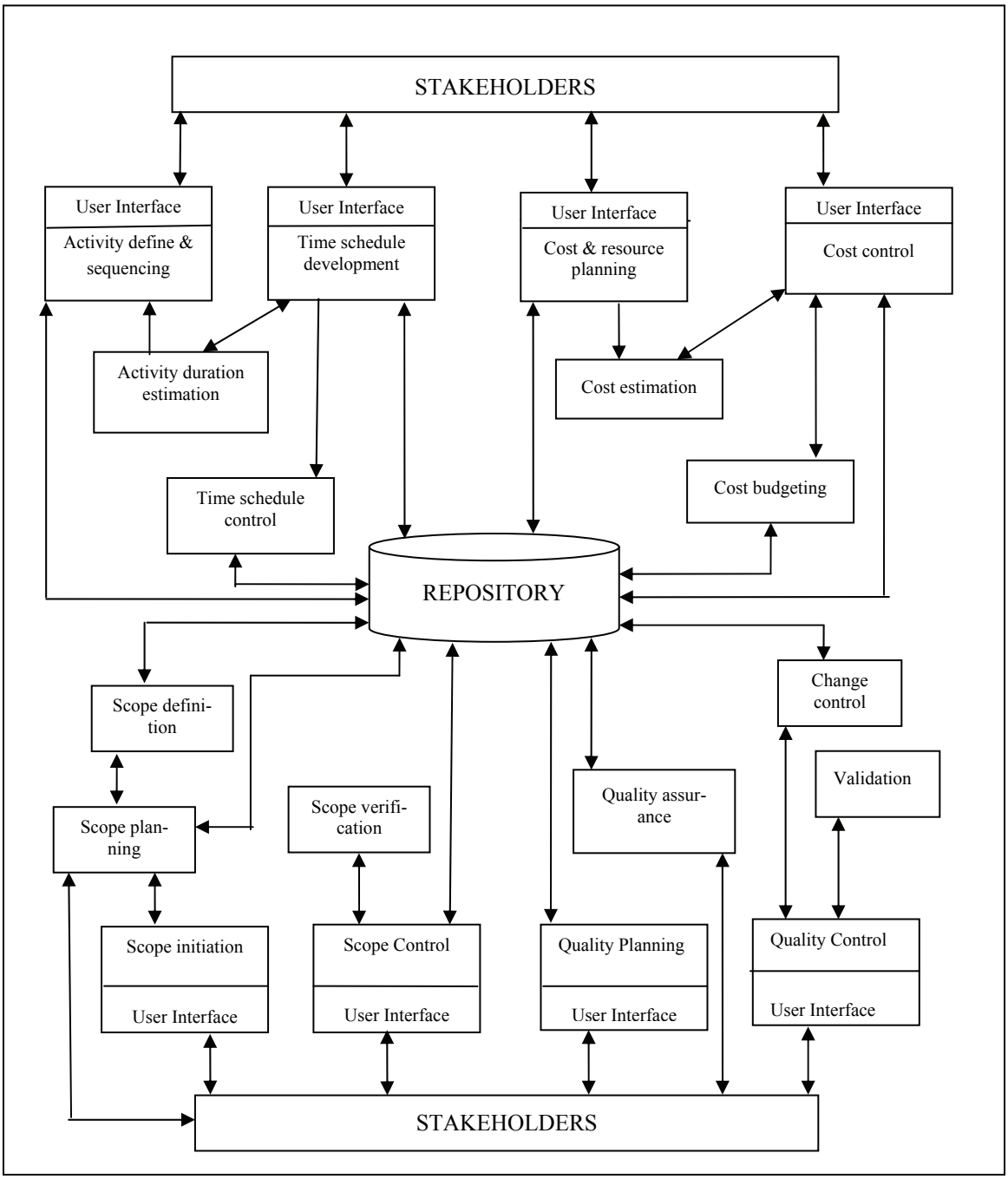


Figure 4: Conceptual model for the core functions: time management, cost management, quality management & scope management

The SPM facilitating functions: communication management, risk management, procurement management and human resource management are depicted in Figure 5.

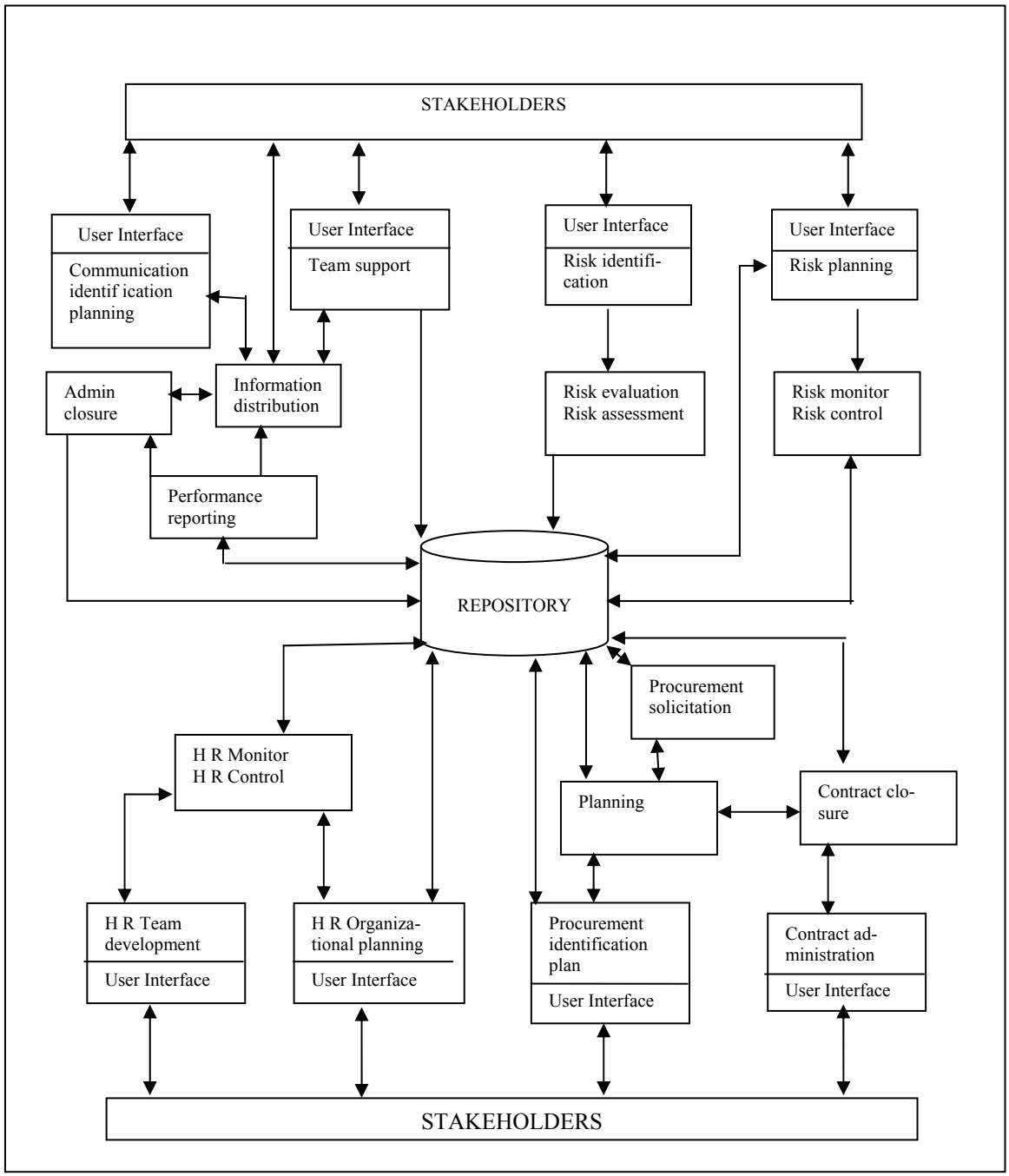


Figure 5: Conceptual model for facilitating functions: communication management, risk management, procurement management and human resource management

We believe that both these models may be implemented as agent black boxes in support of SPM functions.

As prototype of this model one key core function, namely risk management, is currently being implemented in Java and will subsequently be tested. To implement a software agent system an

adaptive and flexible framework is needed that supports multi-agent features that permits the set up of a distributed application, as well as an appropriate level of reasoning capability.

As Java contains most of the required technologies to implement software and mobile agents, such as multithreading, remote method invocation, portable architecture, security features, broadcast support and database connectivity (Wooldridge, 2001), it is viable to implement the system in Java. JADE (Java Agent Development Framework) is a software framework to develop agent-based applications in compliance with the FIPA specifications for interoperable intelligent multi-agent systems. It supports debugging and deployment, the agent platform can be distributed across machines, and the graphical user interface (GUI) can be controlled and changed via a remote GUI. The goal is to simplify the development while ensuring compliance to standards through a comprehensive set of system services and agents. JADE can be considered as agent middleware that implements an agent platform and sustains a development framework. JADE facilitates mobile agent application development, providing key features for distributed network programming. The development and implementation detail, as well as test results, will be detailed in further research.

As part of our research we regard the ISO standards as important guidelines. ISO27001 utilises a model, namely the PDCA cycle to develop and improve an organisation's management system. This cycle was originally designed by Walther Shewart, but revised by the Quality Management authority W Edwards Deming and is currently known as the Plan-Do-Check-Act standard (ISO 17799 Central, 2006). This cycle is used to coordinate continuous improvement efforts, supports daily routine management, supports general problem-solving processes, and also supports SPM, vendor management, human resource management and product development.

Our proposed generic model, as illustrated in Figure 2, is compared to the ISO standard PDCA Cycle in Table 2. The first three phases conforms to that of the PDCA cycle's last three phases. This work will be elaborated on in further research.

Tabel 2: Comparison of PDCA cycle and generic model for software agent frame

PDCA Cycle	Generic model for software agents
	Initiation / evaluation
Plan	Planning
Do	Control / Monitor
Check	Validation / verification
Act	

Conclusion

In this paper we investigated an approach of using software agent technology to address the challenges posed in the Software Project Management (SPM) arena. We focussed on compiling a generic model supporting all key areas of SPM, and designed a generic agent framework to address the common tasks of the key elements. This abstract model was instantiated and detailed to form two comprehensive overall frameworks, supporting all core and facilitating functions. The framework forms a basis for all core and facilitating functions 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 implies that the complexity of creating and maintaining tasks can be greatly reduced. The prototype of this system is currently being implemented for one core function in Java's development platform JADE. 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. Our framework complies with the ISO 27001 standard PDCA cycle, and as such it supports a recognised standard utilised during SPM.

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.

Boehm, B. W. (1989, May). A spiral model of software development and enhancement. *Computer*, 61 -72.

Burstein, M., McDermott, D. & Smith D.R. (2000). Derivation of glue-code for agent interoperation. *Proceedings of the 4th International Conference on Autonomous agents.* ACM Press.

Butte, T. (2002). Technologies for the development of agent-based distributed applications. *Crossroads, 8*(3), 8 – 15.

Croft, D.W. (1997). Intelligent software agents: Definitions and applications. Retrieved May 3, 2003 from http://www.alumni.caltech.edu/~croft/research/agent/definition/

d'Inverno, M. & Luck, M. (2001). *Understanding agent systems.* Berlin: Springer-Verlag.

Dowling, C. (2000), Intelligent agents: some ethical issues and dilemmas. *AICE 2000*. Retrieved August 1, 2003 from http://www.businessit.bf.rmit.edu.au/aice/events/AICE2000/papers/dow.pdf

ELEC 4704 - Software project management. (2003). Department of Electrical and Information Engineering. University of Sydney. Retrieved May 12, 2004 from http://www.ee.usyd.edu.au/elec4704/lec-01.html

Gaeta, M. & Ritrovato, P. (2002). Generalised environment for process management in cooperative software engineering. *Proceedings of the 26th Annual International Computer Software and Applications Conference.* Oxford, England.

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.

Hughes, B. & Cotterel, M. (2006). *Software project management* (4th 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.

ISO 17799 Central. (2006). The A-Z guide for BS7799 and ISO17799 information. Retrieved November 11, 2006, from http://www.17799central.com/pdca.htm

Joslin, D. & Poole, W. (2005). Agent-based simulations for software project planning. *Proceedings of the 2005 Winter Simulation Conference.* IEEE 2005.

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? Retrieved October 12, 2005 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.

Nienaber, R.C. & Barnard, A. (2005). Software quality management supported by software agent technology. *Issues in Informing Science and Information Technology, 2*, 659-670. Available at http://2005papers.iisit.org/I53f40Nien.pdf

Nienaber, R.C. & Cloete, E. (2003). A software agent framework for the support of software project management. *Proceedings of SAICSIT* 2003, pp 16-23

Nienaber, R.C., Cloete, E., & Barnard, A. (2004). Software project risk management supported by agent technology. *The Business Review Journal*, 452-459.

Object Management Group. (2000). Mobile agent facility specification. Retrieved October 8, 2002, from http://www.omg.org

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.

Oghma: Open Source. (2003). Types of software agents. Retrieved May 4, 2003 from http://www.oghma.org

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.

Project Management Institute [PMI]. (2004). *A guide to the project management body of knowledge (PMBOK Guide).*

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.

Schwalbe, K. (2006). *Information technology project management* (4th ed.). Thompson Learning.

The Standish Group. (1995).CHAOS. Retrieved May 4, 2005 from http://www.standishgroup.com/

The Standish Group. (2000). EXTREME CHAOS. Retrieved May 4, 2005 from http://www.standishgroup.com/

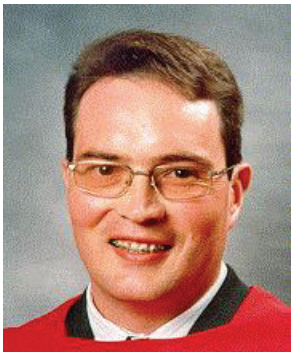
The Standish Group. (2003). Latest Standish Group CHAOS report shows product success rates have improved by 50%. Retrieved March 30, 2005 from http://www.standishgroup.com/

Venners, B. (2000). *Inside the Java virtual machine* (2nd ed.). McGraw-Hill.

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

Biographies

Ms **Rita C Nienaber** is a senior lecturer in the School of Computing at the University of South Africa. 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 modules including 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 School of Computing at the 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, semantic web technologies, computer ethics as well as graph grammar languages.