

A MODEL FOR ENHANCING SOFTWARE PROJECT MANAGEMENT USING SOFTWARE AGENT TECHNOLOGY

By

RITA C. NIENABER

submitted in accordance with the requirements

for the degree of

DOCTOR OF PHILOSOPHY

in the subject

COMPUTER SCIENCE

at the

UNIVERSITY OF SOUTH AFRICA

PROMOTER: PROF. ELMÈ SMITH

CO-PROMOTER: PROF. ANDRIES BARNARD

JUNE 2008

To my husband, Sarel, who incited my interest in computer science - for his unflagging support and encouragement throughout this undertaking.

ACKNOWLEDGEMENTS

To the Almighty and Heavenly Father for supporting me and giving me the strength to persevere with and conclude this study.

I wish to express my gratitude and heartfelt thanks to my promotors:

- Prof. Elmé Smith, for her creative input, guidance and friendship throughout my research project.
- Prof. Andries Barnard who in extremely difficult circumstances still gave his valuable input and insight.

My family – Sarel, Marie-Louise and Ettienne, Karen and Werner, Tanya and Rohan – thank you for all the love, support and understanding.

My grandchildren – Michael, Francois, Carli, Zenia and Arno – to whom I can now give my undivided attention.

The Bit2PhD group led by Prof. Alta van der Merwe – thank you to each of you for your motivation and support.

My friends and colleagues for their support and encouragement.

ABSTRACT

The present study has originated from the realisation that numerous software development projects either do not live up to expectations or fail outright. The scope, environment and implementation of traditional software projects have changed due to various reasons such as globalisation, advances in computing technologies and, last but not least, the development and deployment of software projects in distributed, collaborative and virtual environments. As a result, traditional project management methods cannot and do not address the added complexities found in this ever-changing environment.

In this study the processes and procedures associated with *software project management* (SPM) were explored. SPM can be defined as the process of planning, organising, staffing, monitoring, controlling and leading a software project. The current study is principally aimed at making a contribution to enhancing and supporting SPM.

A thorough investigation into software agent computing resulted in the realisation that software agent technology can be regarded as a new paradigm that may be used to support the SPM processes. *A software agent* is an autonomous system that forms part of an environment, can sense the environment and act on it over a period of time, in pursuit of its own agenda. The software agent can also perceive, reason and act by selecting and executing an appropriate action. The unique requirements of SPM and the ways in which agent technology may address these were subsequently identified. It was concluded that agent technology is specifically suited to geographically distributed systems, large network systems and mobile devices. Agents provide a natural metaphor for support in a team environment where cooperation and the coordination of actions toward a common goal, as well as the monitoring and controlling of actions are strongly supported. Although it became evident that agent technology is indeed being applied to areas and sections of the SPM environment, it is not being applied to the whole spectrum, i.e. *to all core and facilitating functions of SPM*. If software agents were to be used across the whole spectrum of SPM

processes, this could provide a significant advantage to software project managers who are currently using other contemporary methods.

The “SPMSA” model (Software Project Management supported by Software Agents) was therefore proposed. This model aims to enhance SPM by taking into account the unique nature and changing environment of software projects. The SPMSA model is unique as it supports the entire spectrum of SPM functionality, thus supporting and enhancing each key function with a team of software agents. Both the project manager and individual team members will be supported during software project management processes to simplify their tasks, eliminate the complexities, automate actions and enhance coordination and communication. Virtual teamwork, knowledge management, automated workflow management and process and task coordination will also be supported.

A prototype of a section of the risk management key function of the SPMSA model was implemented as ‘proof of concept’. This prototype may be expanded to include the entire SPMSA model and cover all areas of SPM. Finally, the SPMSA model was verified by comparing the SPM phases of the model to the Plan-Do-Check-Act (PDCA) cycle. These phases of the SPMSA model were furthermore compared to the basic phases of software development as prescribed by the ISO 10006:2003 standard for projects. In both cases the SPMSA model compared favourably.

Hence it can be concluded that the SPMSA model makes a fresh contribution to the enhancement of SPM by utilising software agent technology.

TABLE OF CONTENTS

CHAPTER 1	2
1 INTRODUCTION	2
1.1 INTRODUCTION.....	3
1.2 UNIQUE NATURE OF SOFTWARE PROJECT MANAGEMENT	4
1.3 MOTIVATION FOR THIS STUDY	4
1.4 PROBLEMS TO ADDRESSED	5
1.4.1 <i>Do standard SPM practices take into account the unique nature and changing environment of software projects (SPs)?.....</i>	<i>6</i>
1.4.2 <i>How can SPM processes be supported and enhanced in a distributed environment?</i>	<i>6</i>
1.4.3 <i>Has software agent technology been applied to the SPM environment?</i>	<i>7</i>
1.4.4 <i>How can software agent technology be incorporated and utilised by SPM to enhance the entire SPM environment?.....</i>	<i>7</i>
1.5 PUBLICATIONS RESULTING FROM THE STUDY	8
1.6 KEY TERMINOLOGY	9
1.7 OUTLINE OF THE STUDY	10
1.7.1 <i>Part I: Introduction</i>	<i>10</i>
1.7.2 <i>Part II: Theoretical background</i>	<i>11</i>
1.7.3 <i>Part III: The SPMSA model</i>	<i>11</i>
1.7.4 <i>Part IV: Conclusion.....</i>	<i>12</i>
CHAPTER 2	13
2 RESEARCH METHODOLOGY.....	13
2.1 INTRODUCTION.....	14
2.2 RESEARCH METHODOLOGY	14
2.2.1 <i>Research paradigm.....</i>	<i>15</i>
2.2.2 <i>Preliminary literature review.....</i>	<i>16</i>
2.2.3 <i>Research questions.....</i>	<i>16</i>
2.2.4 <i>Research strategies</i>	<i>16</i>
2.2.5 <i>Data generation methods</i>	<i>17</i>
2.2.6 <i>Data analysis methods</i>	<i>19</i>
2.3 SCOPE OF THE STUDY	20
2.4 CONCLUSION	20
CHAPTER 3	23

3	SOFTWARE PROJECT MANAGEMENT	23
3.1	INTRODUCTION.....	24
3.2	UNIQUE CHARACTERISTICS OF SPM.....	25
3.3	CHANGING ENVIRONMENT OF SPM	26
3.4	PROBLEMS WITH SOFTWARE PROJECTS	28
3.4.1	<i>Minimising project failure</i>	<i>30</i>
3.5	SOFTWARE PROJECT MANAGEMENT FRAMEWORK	31
3.6	CORE FUNCTIONS.....	33
3.6.1	<i>Scope management.....</i>	<i>33</i>
3.6.2	<i>Time management</i>	<i>35</i>
3.6.3	<i>Cost management</i>	<i>37</i>
3.6.4	<i>Quality management.....</i>	<i>39</i>
3.7	FACILITATING FUNCTIONS	42
3.7.1	<i>Human resource management.....</i>	<i>42</i>
3.7.2	<i>Communications management</i>	<i>44</i>
3.7.3	<i>Risk management.....</i>	<i>45</i>
3.7.4	<i>Procurement management</i>	<i>47</i>
3.8	CONCLUSION	49
	CHAPTER 4	51
4	SOFTWARE AGENT COMPUTING	51
4.1	INTRODUCTION	52
4.2	AGENT TECHNOLOGY.....	52
4.2.1	<i>Emerging trends as drivers for agent technology.....</i>	<i>54</i>
4.3	WHAT IS A SOFTWARE AGENT?	56
4.3.1	<i>Classes of software agents.....</i>	<i>58</i>
4.4	MOBILE SOFTWARE AGENTS	60
4.4.1	<i>History of mobile agents.....</i>	<i>61</i>
4.4.2	<i>Characteristics</i>	<i>64</i>
4.4.3	<i>Advantages of mobile agents.....</i>	<i>65</i>
4.4.4	<i>Disadvantages of using mobile agents.....</i>	<i>67</i>
4.5	CONCLUSION	70
	CHAPTER 5	74
5	SOFTWARE AGENTS IN SPM.....	74
5.1	INTRODUCTION.....	75

5.2	MOBILE AGENT DEVELOPMENT	75
5.2.1	<i>Mobile agent development methodologies</i>	76
5.3	MOBILE AGENT IMPLEMENTATION.....	79
5.3.1	<i>Agent development environments</i>	80
5.4	APPLICATIONS OF MULTI-AGENT SYSTEMS	84
5.4.1	<i>Agents for Electronic Commerce</i>	85
5.4.2	<i>Agents for Information Retrieval and Management</i>	85
5.4.3	<i>Agents for Network and Internet</i>	86
5.4.4	<i>Agents for workflow and business process management</i>	87
5.4.5	<i>Agents used in project management</i>	87
5.5	CONCLUSION	92
CHAPTER 6	95
6	MODEL – SCOPE AND CONCEPT	95
6.1	INTRODUCTION.....	96
6.2	SCOPE OF THE MODEL	97
6.3	CONCEPT OF THE MODEL	98
6.3.1	<i>Phases of software development for each SPM key function</i>	100
6.3.2	<i>Software agent framework to support each SPM key function</i>	100
6.4	CONCLUSION	103
CHAPTER 7	104
7	THE SPMSA MODEL	104
7.1	INTRODUCTION.....	105
7.2	SOFTWARE PROJECT MANAGEMENT KNOWLEDGE AREAS.....	105
7.2.1	<i>Scope management</i>	107
7.2.2	<i>Time management</i>	110
7.2.3	<i>Cost management</i>	114
7.2.4	<i>Quality management</i>	116
7.2.5	<i>Human resource management</i>	119
7.2.6	<i>Communication management</i>	122
7.2.7	<i>Risk management</i>	126
7.2.8	<i>Procurement management</i>	129
7.3	THE “SOFTWARE PROJECT MANAGEMENT SUPPORTED BY SOFTWARE AGENTS” (SPMSA) MODEL.....	132
7.4	CONCLUSION	141
CHAPTER 8	142

8	PROTOTYPE IMPLEMENTATION	142
8.1	INTRODUCTION.....	143
8.2	DEVELOPING THE PROTOTYPE	143
8.2.1	<i>Requirements analysis phase</i>	<i>144</i>
8.2.2	<i>Design Phase</i>	<i>148</i>
8.3	PROTOTYPE IMPLEMENTATION	149
8.3.1	<i>The Technological Platform.....</i>	<i>149</i>
8.3.2	<i>Overview of the prototype.....</i>	<i>152</i>
8.3.3	<i>Outcome of using JPMPS.....</i>	<i>164</i>
8.4	CONCLUSION	165
	CHAPTER 9	167
9	MODEL VERIFICATION	167
9.1	INTRODUCTION.....	168
9.2	ISO STANDARDS	168
9.3	SPMSA MODEL VERIFICATION.....	170
9.3.1	<i>PDCA cycle.....</i>	<i>170</i>
9.3.2	<i>ISO 10006:2003</i>	<i>173</i>
9.4	CONCLUSION	178
	CHAPTER 10	180
10	CONCLUSION.....	180
10.1	INTRODUCTION	181
10.2	RESEARCH OVERVIEW.....	181
10.2.1	<i>Do standard SPM practices take into account the unique nature and changing environment of software projects (SP)?.....</i>	<i>182</i>
10.2.2	<i>How can SPM processes be supported and enhanced in a distributed environment?</i>	<i>183</i>
10.2.3	<i>Has software agent technology been applied to the SPM environment?.....</i>	<i>184</i>
10.2.4	<i>How can software agent technology be incorporated and utilised by SPM to enhance the entire SPM environment?.....</i>	<i>184</i>
10.3	CONTRIBUTION OF SPMSA MODEL	185
10.4	FUTURE RESEARCH.....	186
11	REFERENCES	188

APPENDIX A	PAPER PRESENTED AT SAICSIT2003, PUBLISHED IN THE CONFERENCE PROCEEDINGS.	A-1
APPENDIX B	PAPER PRESENTED AT THE GLOBAL BUSINESS AND ECONOMIC CONFERENCE, AND PUBLISHED IN THE BUSINESS REVIEW, CAMBRIDGE 2 , (1) AUGUST 2004.	B-1
APPENDIX C	PAPER PUBLISHED IN PROCEEDINGS IN INFORMING SCIENCE AND INFORMATION TECHNOLOGY CONFERENCE INSITE 2006, FLAGSTAFF, ARIZONA, USA.	C-1
APPENDIX D	PAPER PRESENTED AT INSITE 2007 AND PUBLISHED IN THE INTERDISCIPLINARY JOURNAL OF INFORMATION, KNOWLEDGE, KNOWLEDGE AND MANAGEMENT, VOLUME 2, JANUARY 2008.	D-1
APPENDIX E	PAPER PRESENTED AT IADIS, 2008: THE APPLIED COMPUTING INTERNATIONAL CONFERENCE, APRIL, ALGARVE, PORTUGAL. THIS PAPER WAS PUBLISHED IN THE CONFERENCE PROCEEDINGS.	E-1
APPENDIX F	THIS PAPER WAS SUBMITTED TO THE INTERNATIONAL JOURNAL OF INFORMATION MANAGEMENT IN MAY 2008.	F-1

LIST OF FIGURES

FIGURE 2.1	MODEL OF THE RESEARCH METHODOLOGY (ADAPTED FROM OATES, 2006)	14
FIGURE 3.1	SOFTWARE PROJECT MANAGEMENT FRAMEWORK (ADAPTED FROM SCHWALBE, 2006)	32
FIGURE 3.2	SOFTWARE RISK MANAGEMENT (ADAPTED FROM BOEHM, 1991)	46
FIGURE 4.1	THE PERCEIVE-REASON-ACT CYCLE	58
FIGURE 5.1	JADE AGENT PLATFORM	82
FIGURE 6.1	SCOPE OF THE PROPOSED MODEL	97
FIGURE 6.2	CONCEPTUAL VIEW OF THE SPMSA MODEL	99
FIGURE 7.1	SCOPE MANAGEMENT FUNCTION	107
FIGURE 7.2	TIME MANAGEMENT FUNCTION	111
FIGURE 7.3	COST MANAGEMENT FUNCTION	114
FIGURE 7.4	QUALITY MANAGEMENT FUNCTION	117
FIGURE 7.5	HUMAN RESOURCE MANAGEMENT FUNCTION	120
FIGURE 7.6	COMMUNICATION MANAGEMENT FUNCTION	122
FIGURE 7.7	RISK MANAGEMENT FUNCTION	126
FIGURE 7.8	PROCUREMENT MANAGEMENT FUNCTION	130
FIGURE 7.9	THE SPMSA MODEL - CORE FUNCTIONS	135
FIGURE 7.10	THE SPMSA MODEL - FACILITATING FUNCTIONS	136
FIGURE 8.1	USE-CASE FOR THE JADE PROJECT MANAGEMENT PROTOTYPE SYSTEM (JPMPS)	146
FIGURE 8.2	SOCIAL AGENT MODEL FOR THE JPMPS	147
FIGURE 8.3	AGENT MANAGEMENT SYSTEM	150
FIGURE 8.4	DIRECTORY FACILITATOR	152
FIGURE 8.5	INPUT SCREEN FOR THE CSWA PROJECT	154
FIGURE 8.6	TASK: DEPLOY SENSOR WEB APPLICATION	155
FIGURE 8.7	PERSONAL ASSISTANT AGENT: WABO	156
FIGURE 8.8	RISK PROBABILITY AND MONETARY VALUE INPUT: EXAMPLE 1	157
FIGURE 8.9	RISK PROBABILITY AND MONETARY VALUE INPUT: EXAMPLE 2	158
FIGURE 8.10	TASK COMPLETION INPUT SCREEN	159
FIGURE 8.11	RISK AGENT IN CONTAINER 1	160
FIGURE 8.12	RISK AGENT IN CONTAINER 2	160
FIGURE 8.13	PERSONAL ASSISTANT AGENT: TERENCE	161
FIGURE 8.14	RISK PRIORITISATION REPORT SCREEN	162
FIGURE 8.15	TEAM LEADER REPORT	163
FIGURE 9.1	THE PDCA CYCLE	171

LIST OF TABLES

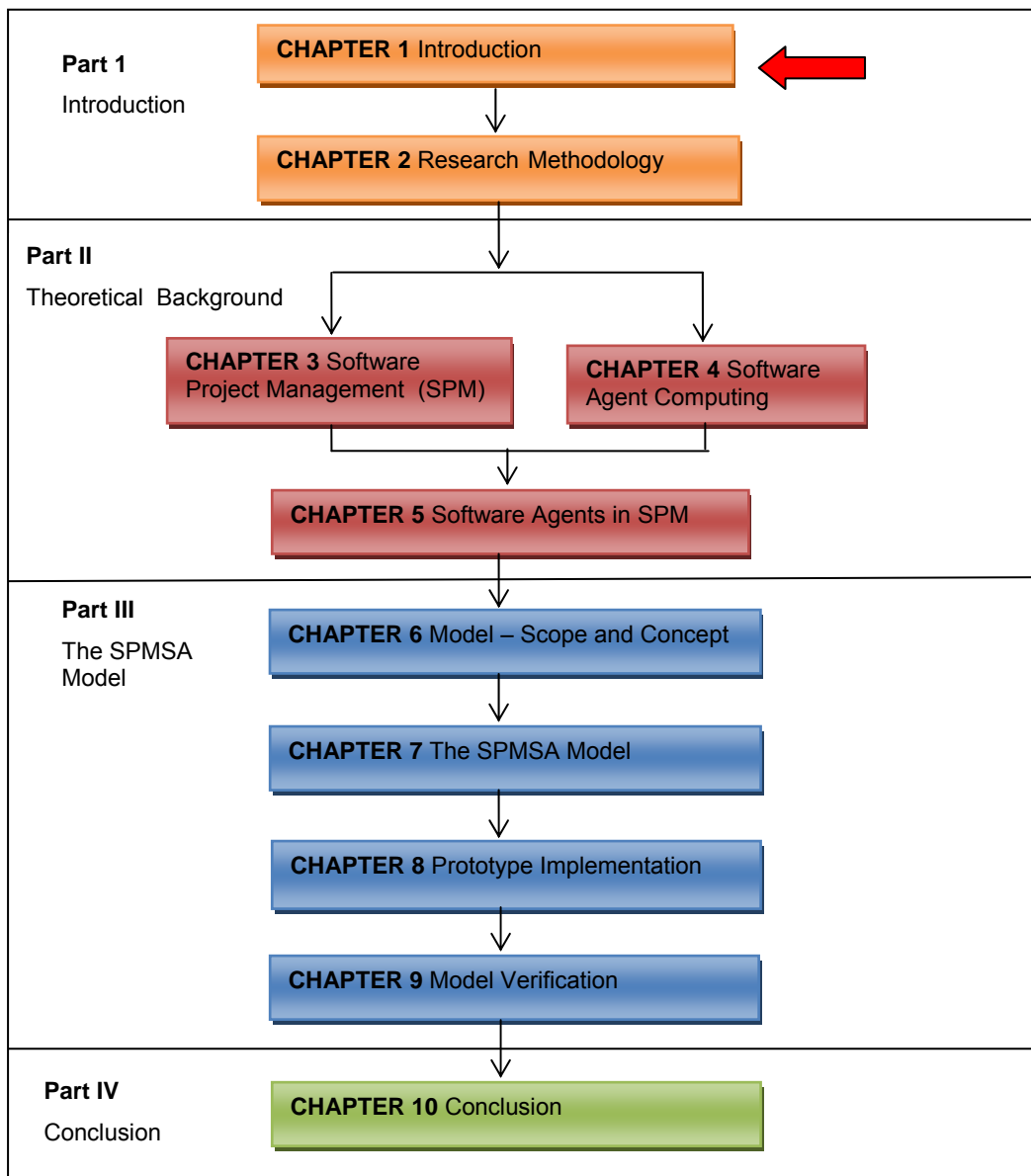
TABLE 2.1	RESEARCH QUESTIONS	16
TABLE 2.2	RESEARCH METHODOLOGY	20
TABLE 3.1	THE STANDISH GROUP REPORT.....	30
TABLE 3.2	PHASES OF THE KNOWLEDGE AREAS OF SPM	49
TABLE 4.1	AGENT PROPERTIES	61
TABLE 4.2	MOBILE AGENT CHARACTERISTICS	65
TABLE 4.3	CHALLENGES REGARDING AGENT TECHNOLOGY (LUCK ET AL., 2005).....	68
TABLE 4.4	SPM FEATURES TO BE ADDRESSED BY AGENT TECHNOLOGY.....	71
TABLE 5.1	PLATFORMS FOR AGENT DEVELOPMENT.....	83
TABLE 7.1	AGENTS AND THEIR TASKS	133
TABLE 7.2	LIMITATIONS OF SPM ADDRESSED BY AGENT TECHNOLOGY.....	137
TABLE 8.1	AGENT INTERACTION WITH OTHER AGENTS	148
TABLE 9.1	PDCA CYCLE VS SPMSA MODEL	172
TABLE 9.2	PDCA CYCLE VS GENERIC PHASES OF THE SPMSA MODEL	173
TABLE 9.3	SPMSA MODEL VS ISO 10006:2003.....	174
TABLE 9.4	ISO 10006:2003 CLAUSES NOT REFLECTED IN THE SPMSA MODEL.....	176
TABLE 9.5	SPMSA PROCESSES NOT REFLECTED IN ISO 10006:2003.....	177

PART I

INTRODUCTION

CHAPTER 1

1 INTRODUCTION



1.1 INTRODUCTION

Most current business undertakings are supported by software applications. The quality, effectiveness and efficiency of these applications determine the success or failure of many business solutions. As a result, businesses often find that they need to obtain a competitive advantage through the development of software projects that support crucial business activities. The quality of the software development process plays a key role in the quality of the software implementation. Improvements in the development of project management software can therefore result in a significant improvement in software quality (Schwalbe, 2006).

Numerous software development projects either do not live up to expectations or they fail outright. This is clear from the fact that software projects often do not comply with the traditional standard measurements of success, namely time, cost and scope (Schwalbe, 2006). For example, Marchewka (2003) reports that of the more than \$250 billion that the United States spent on IT projects, 31% were cancelled before completion. Only 53% were completed but they had exceeded their budgets and time schedules and were not compliant with the specifications. This explains why researchers and practitioners are continuously trying to find new, and enhance existing solutions to these problems (Boehm, 1991; Chen, Nunamaker, Romano and Briggs, 2003; Marchewka, 2003; The Standish Group, 2005).

In some initial attempts to address problems associated with software development, traditional *project management* (PM) techniques were applied to the development of software projects. However, over time project management methods seemed to lack the ability to address the unique characteristics of the software development domain (Olson, 2004; Hughes and Cotterell, 2006). This led to the development of Software Project Management as an independent application area and field of study (Romano, Chen and Nunamaker, 2002; Chen et al., 2003).

1.2 UNIQUE NATURE OF SOFTWARE PROJECT MANAGEMENT

Software Project Management (SPM) differs from General Project Management as certain inherent characteristics are unique to software development (Brooks, as quoted by Hughes and Cotterell, 2006). These characteristics are invisibility, complexity, conformity and flexibility.

- *Invisibility* implies that the process of developing the software cannot be seen (is not visual); thus it is difficult to control, monitor, measure and estimate project progress.
- *Complexity* of software projects is increased in that software projects include not only the development, but also the implementation and maintenance of a system that may be distributed and that interfaces with many existing systems.
- *Conformity* of software is essential. Traditional disciplines involve physical non-changing resources, whereas software projects involve a variety of resources where the software is expected to conform to the requirements of humans and organisations.
- *Flexibility* is needed as software systems are required to conform to the standards of the organisation. Thus it is subject to a high degree of change.

The above unique factors contribute to shortcomings in the field of SPM and therefore need attention.

1.3 MOTIVATION FOR THIS STUDY

The available literature in this field reveals that ongoing research is conducted to address the current shortcomings in the management of software projects (Addison and Vallabh, 2002; Roy, 2004; Sherer, 2004; The Standish Group, 2003). Practitioners have attempted to apply several Software Engineering principles to different Software Project Management processes (Lethbridge and Laganieri, 2001). They have also explored standard structured analysis and design methods and incorporated object-oriented approaches to overcome the aforementioned

shortcomings (Gelbard, Pliskin and Spiegler, 2002; Hughes and Cotterell, 2006). Different standard project management approaches exist, which are applicable to different areas of software project management, such as PRINCE 2 and BS 6079:1996 (Hughes and Cotterell, 2006). Yet many software projects still fail to comply with the *triple constraints* of time, cost and scope (Oghma: Open Source, 2003).

The problems mentioned above can be ascribed to various factors, the most important of which is the fact that the SPM environment has changed dramatically over the past decade and is still changing rapidly due to globalisation and advances in computing technology (Romano et al., 2002; Zanoni and Audy, 2003). The traditional single project, which was commonly executed at a single location, has evolved into distributed, collaborative projects deployed in distributed and collaborative environments. This means that traditional project management methods cannot and do not address the added complexities found in a distributed environment, such as efficient task scheduling, tracking and monitoring, as well as the effective sharing of information and knowledge among project contributors. There is therefore an urgent need for managing software project risks in such a way that this complex distributed environment is addressed and optimally supported.

1.4 PROBLEMS TO BE ADDRESSED

With the advent of global enterprises and virtual organisations, the environment impacting on traditional software project management has changed. Outsourcing of projects has become commonplace and adds to the complexity of managing the project management process. It furthermore implies that traditional project management methods are unable to address the added complexities found in a distributed environment. Consequently, tools are required for the effective sharing of information and knowledge among project contributors, as well as for efficient task scheduling, tracking and monitoring. High levels of collaboration, task interdependence and distribution have become essential across time, space and technology (Chen et al., 2003). This thesis will therefore explore the processes and procedures associated with SPM in an attempt to contribute to the enhancement of

SPM. It will accordingly propose a SPM model that enhances SPM processes by incorporating a software agent technology framework. The main issues to be addressed in this thesis are consequently defined in the form of the following research questions:

1.4.1 Do standard SPM practices take into account the unique nature and changing environment of software projects (SPs)?

Over the past years the SPM environment has changed and it is still evolving and developing. Standard SPM practices focus on a single project, commonly executed at a single location with localised team members. This has changed due to factors such as globalisation and advances in computing technology (Romano et al., 2002; Zanoni and Audy, 2003). Nowadays projects are of a distributed and collaborative nature, and they are deployed in distributed and collaborative environments. SPM is also characterised by its unique and dynamically changing nature, which differs greatly from standard project management.

Innovative SPM practices should take full cognisance of this unique nature and changing environment of SPM. An attempt will be made to determine whether traditional SPM methods adequately address the added complexities found in a distributed environment. This thesis will explore the changing environment and unique nature of SPM as well as the processes and procedures associated with SPM, in an effort to address this question.

1.4.2 How can SPM processes be supported and enhanced in a distributed environment?

Software that supports crucial business activities may be utilised to gain a competitive advantage for its organisation. In other words, the quality of the software development process, as well as improvements in the development of project management software can significantly enhance the quality of the software (Schwalbe, 2006).

Since the operational environment of SPM has changed, new methods are needed to enhance and support standard SPM practices. Different paradigms are evolving and several may hold promise to address both this changing environment and the unique nature of SPM. Several paradigms such as software engineering principles, agile methodologies and structured analysis and design principles have been applied in an effort to improve SPM practices, though not with much success.

The questions that arise are whether a new paradigm could minimise the problems with current practices, and whether another paradigm could support the intrinsic aspects that cause the failure of projects. The researcher plans to investigate the possibility of using software agent technology to address SPM problems in a distributed environment in order to enhance SPM processes.

1.4.3 Has software agent technology been applied to the SPM environment?

Various software agent applications have been developed over the past few years. The researcher aims to investigate whether software agent technology has been applied to support any processes in the SPM environment and/or to address SPM problems in a distributed environment. In addition, she wishes to find out whether software agent technology has been used to support or enhance single functions or larger application areas in the SPM environment.

1.4.4 How can software agent technology be incorporated and utilised by SPM to enhance the entire SPM environment?

To address this question, the researcher will implement an SPM model entitled SPMSA (Software Project Management supported by Software Agents) that she developed for this purpose. The SPMSA model enhances and supports SPM processes by incorporating a software agent framework.

1.5 PUBLICATIONS RESULTING FROM THE STUDY

The following peer-reviewed publications were generated as result of the research conducted for this thesis (see Appendix A to F for the articles):

- 1) Nienaber R.C. and Cloete E. 2003. **A Software Agent Framework for the support of Software Project Management.** In Proceedings of IT Research in Developing Countries, Midrand, Gauteng. (SAICSIT 2003). ISBN: 1-58113-774-5, pp. 16-23.
- 2) Nienaber R.C., Cloete E. and Barnard A. 2004. **Software Project Risk Management Supported by Agent Technology.** In Proceedings of the Global Business and Economic Conference, August 2004, Istanbul, Turkey. *The Business Review*, Cambridge, **2**, (1) 2004. ISSN 1540 -1200, pp. 452-459.
- 3) Nienaber R.C. and Barnard A. 2006. **Software Quality Management supported by Software Agent Technology.** In Proceedings in Informing Science and Information Technology Conference (INSITE 2006), Flagstaff, Arizona, USA. ISSN: 1547-5840, pp. 659-669.
- 4) Nienaber R.C. and Barnard A. 2007. **A Generic Agent Framework to Support Various Software Project Management Processes.** In the proceedings of the conference on Issues in Informing Science and Information Technology (INSITE 2007), June 22-25 2007, Ljubljana, Slovenia. This paper received the “best paper” award, and was accordingly published in the *Interdisciplinary Journal of Information, Knowledge and Management*, Volume 2, pp. 149-162, January 2008.
- 5) Nienaber R. C., Smith E., Barnard A., and Van Zyl T., 2008. **Software Agent Technology supporting Risk Management in SPM.** In the conference proceedings of IADIS International Conference Applied Computing (IADIS2008), April 10 – 13, 2008, Algarve, Portugal.
- 6) Nienaber R.C. and Smith E. 2008. **Enhancing and supporting SPM: The Software Project Management supported by Software Agents model.** Submitted to the International Journal of Information Management (2008). *Pending.*

Articles 1 to 5 were presented at international conferences.

Article 4 was also published in an international journal.

Article 6 has recently been submitted to an accredited international journal.

1.6 KEY TERMINOLOGY

Having thoroughly studied the available literature, the researcher found that numerous definitions and meanings exist with respect to software project management terms and concepts, as well as concepts related to software agents (Marchewka, 2003; Olson, 2004; Schwalbe, 2006; Hughes and Cotterell, 2006). For the purpose of this thesis, the following terminology applies and can be used as a frame of reference for further discussions in this thesis.

An *agent* is a system that is situated within a part of an environment and that senses that environment and acts on it, over time, in pursuit of its own agenda and so as to effect what it senses in the future (Franklin and Graesser, 1996; Wooldridge, 2002).

A *project* is a temporary endeavour undertaken to accomplish a unique purpose. A project furthermore requires resources, has a primary stakeholder or customer and involves uncertainty (Schwalbe, 2006).

Project management is the application of knowledge, skills, tools and techniques to project activities in order to meet or exceed stakeholder needs and expectations with regard to a project (Elec 4704, 2003).

Software Project Management is the process of planning, organising, staffing, monitoring, controlling and leading a software project (IEEE Standards Board, 1997).

A *software agent* is an autonomous system that forms part of an environment when situated within the said environment. The software agent can sense the environment and act on it over a period of time, in pursuit of its own agenda. The software agent can also perceive by receiving stimuli from its environment, reason by combining

newly acquired information with its existing goals and knowledge, and act by selecting and executing an appropriate action (Franklin and Graesser, 1996).

A mobile agent is an active entity that can migrate autonomously from one location to another (as opposed to a stationary agent), and resume execution at a remote site to perform a task on behalf of its user. It is able to act independently, observe its environment and to adapt to changes in the environment (Dale 1997; Kotz and Gray, 1999; Schoeman, 2005).

An intelligent agent will to a smaller or larger degree contain capabilities of reactivity (implying that it is able to perceive its environment and respond to it in timely fashion), proactivity (entailing exhibiting goal-directed behaviour by taking the initiative to satisfy its design objectives), and a social ability (meaning that it is able to interact and communicate with other agents to satisfy its design objectives) (Wooldridge, 2002).

A mobile agent environment is a software agent system that is distributed over a network of computers. Its primary task is to provide an environment in which agents can execute. The mobile agent environment provides support services for agent movement, connection to environments in which the agent environment exists, as well as services to communication (Green, Hurst, Nangle, Cunningham, Somers and Evans, 1997).

1.7 OUTLINE OF THE STUDY

This thesis consists of four parts, with each part comprising one or more chapters. The figure on page 2 graphically depicts the layout of and relationship between the various chapters. This section describes the outline of the chapters.

1.7.1 Part I: Introduction

Part I, comprising chapters 1 and 2, serves as an introduction and background to this study. **Chapter 1** introduces the field of study, gives a rationale for the current study and highlights the problems to be addressed in the form of research questions.

Chapter 2 of Part I presents the research methodology followed throughout this thesis.

1.7.2 Part II: Theoretical background

Part II serves as theoretical background to this study, and includes chapters 3, 4 and 5. **Chapter 3** of Part II is devoted to a literature study relevant to the area of SPM. This chapter explores the changing and unique nature of the SPM environment. Software project characteristics, as well as its core and facilitating functions are investigated and reported on. The conclusion reached, namely that failure in many areas of SPM points to shortcomings in standard SPM practices, is deemed important as it underpins the need for a new approach to support and enhance standard SPM practices. In **Chapter 4** software agent technology is investigated. Trends driving software agent technology are revealed, and concepts and identifying features of agent technology are discussed. In doing so, salient features of software agent technology are uncovered which motivates the use of such technology. It becomes clear that these characteristics of software agent technology can indeed address the unique and complex requirements of SPM.

In **Chapter 5** the process of agent development is scrutinised. The chapter also explores the utilisation of agent technology in the SPM field and investigates various applications. It transpires that although agent technology has been utilised for certain functions in the SPM environment, it has not been applied to the entire SPM environment.

1.7.3 Part III: The SPMSA model

Part III consists of chapters 6, 7, 8 and 9. It presents the SPMSA (Software Project Management supported by Software Agents) model, which involves SPM processes that are supported by agent technology. **Chapter 6** commences with a presentation of the scope of this model, comprising the main research areas addressed in this thesis, namely SPM and software agent technology. A conceptual view of the model is compiled next to illustrate the generic phases of software development for each SPM

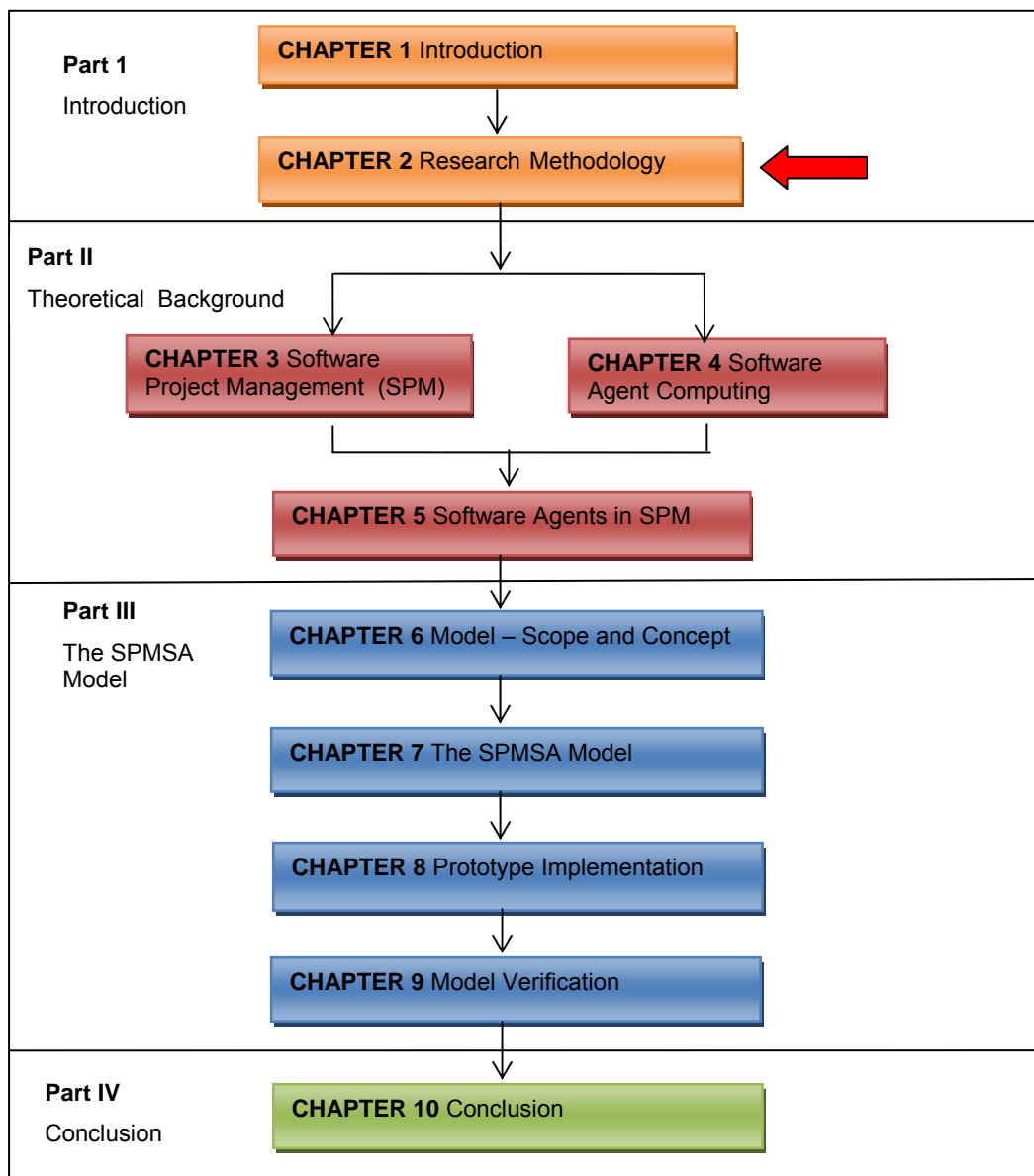
key function, as well as the software agent framework that will address each key area of SPM. It is thus established that the proposed model is specifically tailored to support the unique and changing SPM environment. In **Chapter 7** each of the key areas of SPM is scrutinised and elaborated on. The aim of this section is to compile a comprehensive model – the SPMSA model – to enhance and support the entire SPM environment. **Chapter 8** illustrates the implementation of a section of the SPMSA model as proof of concept in the form of a prototype and explains how this prototype was tested in a real-life scenario. **Chapter 9** substantiates the relevance of the model by comparing it to the Plan-Do-Check-Act (PDCA) cycle, as well as to the ISO 10006:2003 standard.

1.7.4 Part IV: Conclusion

The thesis culminates in **Part IV**, which comprises of a single chapter. **Chapter 10** provides a summary of and reports on the outcomes of the research project. The thesis is concluded with a reflection on areas of further research.

CHAPTER 2

2 RESEARCH METHODOLOGY



2.1 INTRODUCTION

Different research paradigms, models and strategies based on various philosophical foundations and conceptions of reality may be utilised to direct the research process. Chapter 2 delineates the research methodology followed in this thesis, namely that recommended by Oates (2006).

2.2 RESEARCH METHODOLOGY

The purpose of this study is to enhance SPM, and consequently the use of software agent computing as a potential tool to support project managers and role-players during SPM processes is investigated. An overview of the *complete* research process as recommended by Oates (2006) is graphically depicted in Figure 2.1.

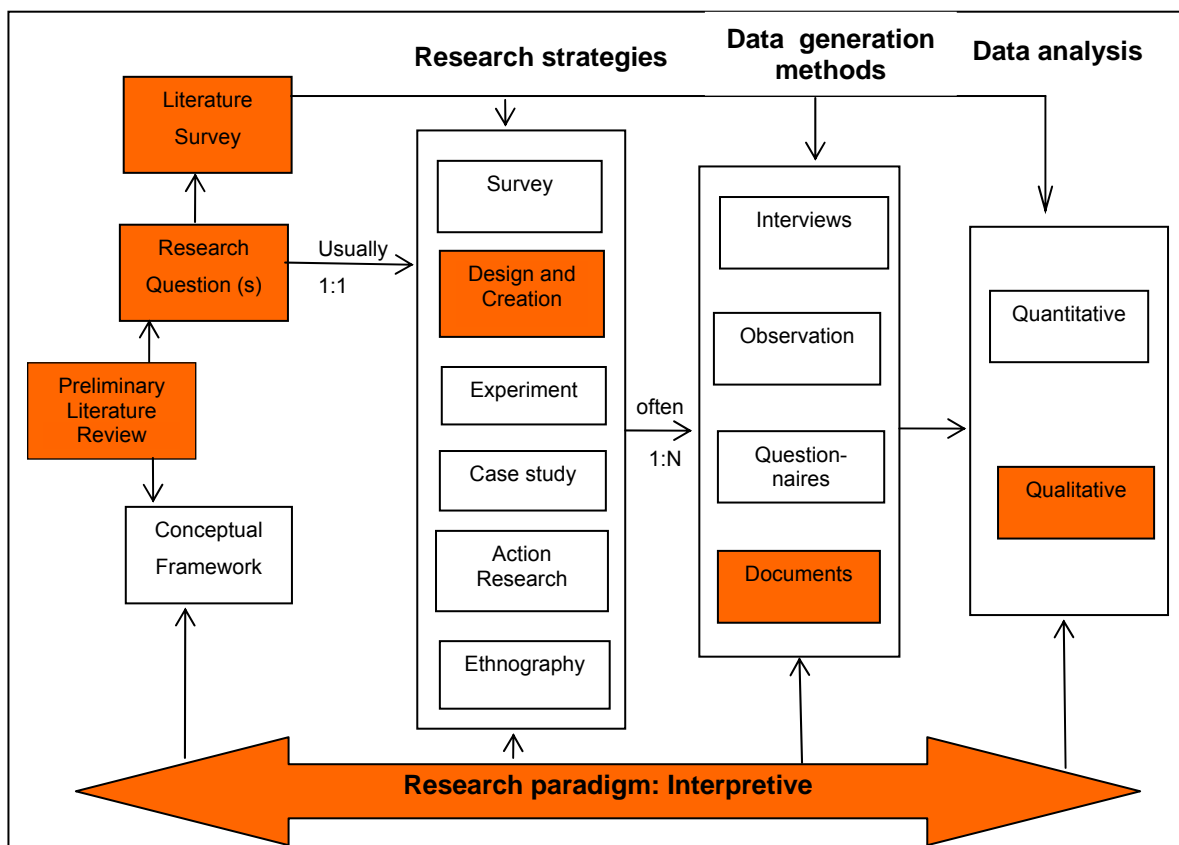


Figure 2.1 Model of the research methodology (adapted from Oates, 2006)

The coloured blocks indicate the specific research methodology steps that were followed in the course of this study.

2.2.1 Research paradigm

A paradigm is defined by Oates (2006) as:

“ a pattern or model or shared way of thinking”.

Various philosophical foundations and concepts of reality influence the ground rules and building blocks on which research paradigms are based and the specific methods that are used by them (de Villiers, 2005; Olivier, 2004). Each paradigm is implemented using associated methodological approaches and strategies. Different philosophical paradigms have different views on the nature of our world (*ontology*) and the methods we use to acquire knowledge about it (*epistemology*). Based on the available literature, three primary research paradigms are identified, namely *positivist*, *interpretive* and *critical research* (Myers, 2006; Oates, 2006). For the purposes of this study, the interpretive research paradigm is the most suitable.

Interpretive research in computing is defined by Oates (2006), in that it:

“concerns itself with understanding the social context of an information system; the social processes by which it is developed and construed by people and through which it influences, and is influenced by its social setting”.

As this study reflects on the practices and functions of SPM, it is inherently a study of processes and interactions. It considers the social context of an information system – specifically the social processes by which it is developed and construed – and thus emphasises the interpretative nature of the study, according to Oates’s criteria.

In recent years interpretive research has become accepted in Information Systems (IS) (Klein and Meyers, 1999; Roode, 2003). According to Klein and Myers (1999) interpretive studies can provide clear insight into IS management and development. Interpretivism typically, but not exclusively, tests research questions and lends itself to qualitative studies (de Villiers, 2005). The interpretive research paradigm underpins the research methodology followed in this study (see Figure 2.1).

2.2.2 Preliminary literature review

The first step in the research methodology followed constitutes a preliminary literature review (see Figure 2.1). The purpose of such a review is to determine what research has so far been done in the specific field of study, and to identify current problems and areas for future research. Following this review, a number of research questions will be formulated. This will be followed by an ongoing literature survey as the research progresses.

2.2.3 Research questions

Having conducted a preliminary literature review concerning SPM, the following research questions were identified (see Chapter 1):

Table 2.1 Research Questions

	Research questions
1	Do standard SPM practices take into account the unique nature and changing environment of software projects?
2	How can SPM processes be supported and enhanced in a distributed environment?
3	Has software agent technology been applied to the SPM environment?
4	How can software agent technology be incorporated and utilised by SPM to enhance the entire SPM environment?

Each chapter in this thesis is devoted to answering one or some part of the research questions above.

2.2.4 Research strategies

The third step in the research methodology involves the research strategies to be executed to address the research questions stated above. *Research strategies* can be described as the approach to answering the research question(s). Oates (2006) identifies various strategies, namely *survey, design and creation, experimenting,*

case studies, action research and ethnography, as depicted in Figure 2.1. The specific approach adopted for this thesis is the design and creation research strategy.

The design and creation research strategy focuses on developing new IT products such as constructs, models, methods and instantiations (March and Smith, 1995).

Quoting Oates (2006), “[a] researcher following the design and creation strategy could offer a construct, model, method or instantiation as a contribution of knowledge”.

Research that utilises the design and creation strategy must illustrate how this research differs from technical development by using concepts of analysis, explanation, argument, justification and critical evaluation (Oates, 2006). A model should be compiled and evaluated. According to Oates (2006), it is rare for any implementation of such a model developed through design and creation research to be a full-blown system that can be used immediately without any other researcher involvement. Instead, the role of implementing the model is that of a prototype to illustrate ideas and constructs, models and methods by which effective and efficient workings systems (involving people and technology) might be achieved. No evaluation of the system in use is necessarily provided (Oates, 2006).

This thesis thus reflects on *design and creation* research as the SPM environment is studied and its core and facilitating functions are defined and compiled into a model of the entire SPM functional area. A software agent framework is then constructed to support the SPM area and a section of this model is subsequently instantiated through a prototype. The model is evaluated by being compared to the PDCA (Plan-Do-Check-Act cycle) as well as to the ISO standard 10006:2003.

2.2.5 Data generation methods

The next step in the research methodology, as illustrated in Figure 2.1, is *data generation*. Data generation includes interviews, observations, questionnaires and documents. The approach followed was adopted from the guidelines set by Oates (2006) and Miles and Huberman (1994). The primary source of data for this thesis is documents that were obtained from formal academic sources. Such documents were obtained from electronic libraries within Computer Science and Information Systems, notably from the ACM, the IEEE (Computer Society Library), Elsevier, ScienceDirect and Springerlink. This primary source was supplemented by verified informal sources such as specific Web sites concerned with software agent technology. Completeness of data or documents was attained through data saturation. Data set saturation was achieved by exploring the reference list of significant and relevant publications. This process was continued until saturation point was reached.

In order to prevent the researcher from being swamped by the amount of data, Oates (2006) recommends techniques to manage and analyse data, namely *data preparation, data reduction, data analysis and interpretation (evaluation)*.

Data preparation involves structuring the data into a format ready for analysis. Similar formats are easier to analyse, such as A4 pages, or computer files. Filing will also benefit by an identified similar format. In this thesis, data preparation was executed and categories were set based on the various functionalities of the application domain studied. These categories were consequently refined. Documents, books, texts and publications were used within the researcher's impressions and experience of, for example, constructing a model or a framework.

Data reduction involves the identification of broad themes within the research topic. Relevant data can be further categorised and ordered by identifying broad categories and units to be refined later. Categories may be identified based on a *deductive approach*, where existing theories are used as base and extended or

expanded, or on an *inductive approach* where categories are identified based purely on data explored. Themes can now be identified and interconnections established. The data reduction process as described by Oates (2006) was followed where referenced publications were selected based on the application domain, namely software project management core and facilitating functions. Categorisation was based on these areas and sub-categories were identified. The journal articles were also analysed, which led to an additional selection of documents.

Data analysis involves the “breaking up” of data into manageable themes, patterns, trends and relationships (Mouton, 2001). Data is analysed to get a clear understanding of the various elements of the data. Through inspecting the data, relationships may be defined between concepts, constructs or variables. The aim of data analysis is to identify or isolate any clear trends, patterns or even themes in the data.

Data interpretation (evaluation) involves the synthesis of one’s data, based on identified trends, into larger coherent structures (Mouton, 2001). The aim of data interpretation is to formulate theories or hypotheses that reflect on the observed patterns or trends in data. Through interpretation of one’s data, the results and findings may be related to existing theoretical frameworks or models.

2.2.6 Data analysis methods

Data can be analysed qualitatively or quantitatively. *Quantitative data analysis* implies data or evidence based on numbers, and is typically generated through experiments and surveys. *Qualitative data analysis* on the other hand, includes non-numerical data such as words, images and items found in researchers’ notes, diaries, documents and tapes. Qualitative studies investigate the *why* and *how* of decision making, as compared to the *what*, *where* and *when* of quantitative studies.

In this study, which was mainly guided by the researcher’s impressions and experience, documents, books, texts and publications were used to construct a

model, agent framework and prototype. The study is therefore qualitative and not quantitative.

2.3 SCOPE OF THE STUDY

This study explores the entire spectrum of Software Project Management as a discipline. The aim of the study is to enhance SPM processes and therefore all core and facilitating functions of SPM are investigated to identify methods or structures to support and enhance the said functions.

Agent taxonomies comprise biological, robotical and computational agents, with differentiation of computational agents into artificial life agents and software agents (Franklin and Graesser, 1996). The main focus of this thesis is on software agents, as will be discussed in Chapter 4.

2.4 CONCLUSION

The research methodology in this thesis, as executed in accordance with the research process of Oates (2006), is summarised in Table 2.2 below.

Table 2.2 Research methodology

Research methodology	This thesis
Research paradigm	Interpretivist
Research strategy	Design and Create
Data generation method	Documents
Data analysis method	Qualitative

Chapter 2 concludes Part I of the thesis. It aimed to describe the area under investigation, formulate the research questions, and consequently elaborate on the research methodology followed to answer the set of research questions. It also served to place the research project in context for the reader.

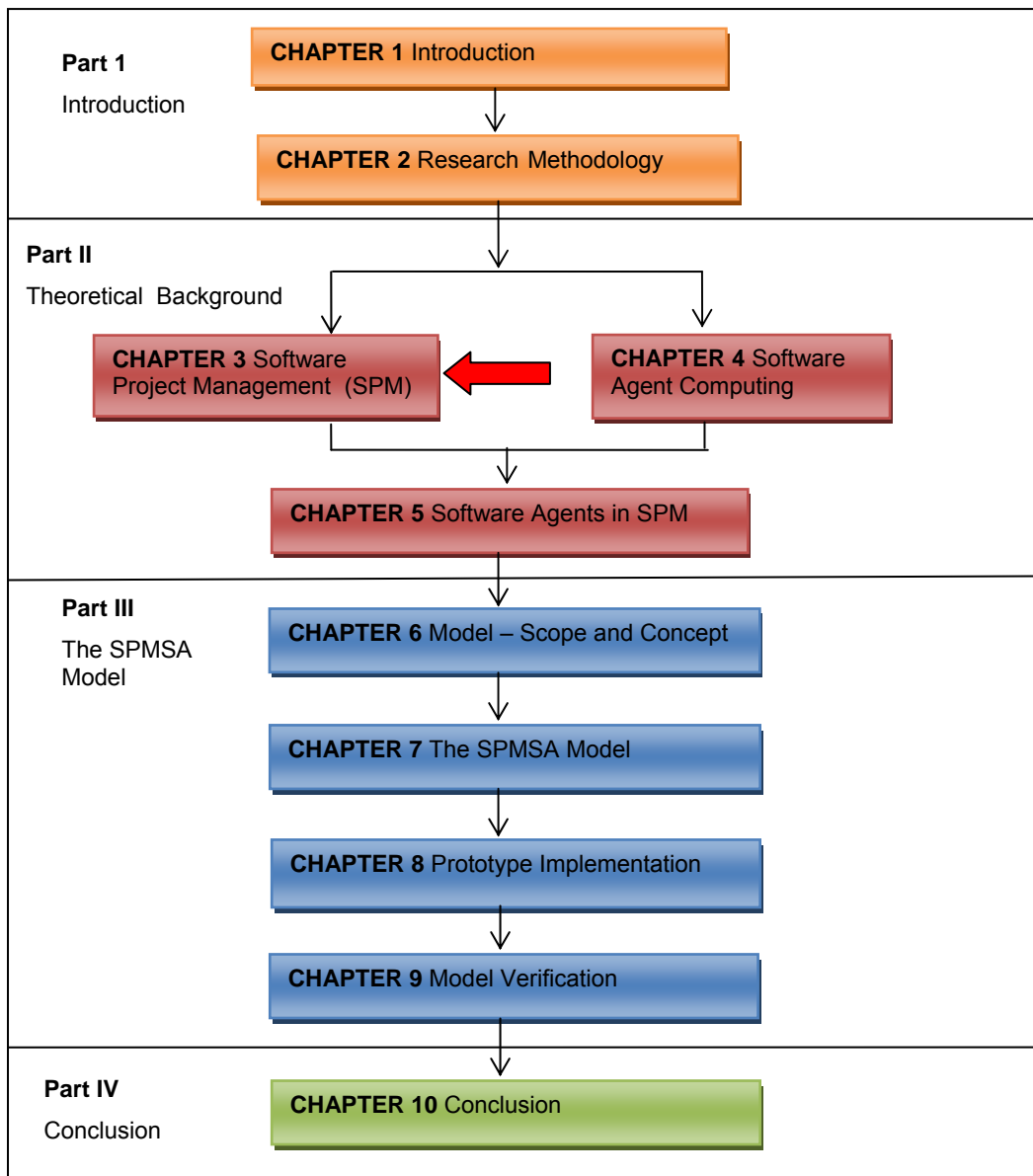
In Part II, the theoretical background of SPM and agent computing will be explored. Essential aspects of SPM, including its unique factors and changing environment, will be investigated. Software agent technology will also be explored to determine whether it may be utilised to support SPM processes.

PART II

THEORETICAL BACKGROUND

CHAPTER 3

3 SOFTWARE PROJECT MANAGEMENT



3.1 INTRODUCTION

Project management is one of the most critical processes for implementing multidisciplinary ventures and leveraging company resources (Thamhain, 2003). Advances in information technology have had a profound impact on project management as an enabling tool, affecting project management capabilities and functioning. This has resulted in a contemporary framework for multi-functional linkages and communication, essential for integrating today's complex projects and project environments. However, in spite of technological advances, the increased level of interconnectivity, distribution and processing, also creates vast challenges involving a wide spectrum of software-related activity management and organisational issues. In fact, complexities and risks of software project development continue to increase (Marchewka, 2003).

Over the past years, the development of software projects have regularly failed to meet user expectations, were commonly delivered late, and mostly exceeded the set budget. Much of this still holds true today, which is why these issues have to be addressed in concrete terms (Chen, et al., 2003; Chen, Lin, Blocker and Cokins, 2005). As a result, the field of SPM is receiving increasing attention and various methods and techniques are utilised to optimise the SPM processes. SPM involves the management of all aspects and issues that are involved in the development of a software project, namely: identification of scope and objectives; 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 (Hughes and Cotterell, 2006). The specific purpose of this chapter is to explore the changing environment of SPM. Software project characteristics, as well as the core and facilitating functions of SPM, are delineated to identify problems experienced during software project management processes.

3.2 UNIQUE CHARACTERISTICS OF SPM

SPM processes comprise their own unique features. Research was conducted by Brooks to emphasise the unique nature of SPM, and reported on in his much cited work *No Silver Bullet* in 1987. In Section 1.2 (Chapter 1) the unique nature of SPM was outlined and characteristics unique to software projects were listed as invisibility, complexity, conformity and flexibility. These aspects contribute to the difficult task of pinpointing a software project as an exact task with a specific beginning, end and deliverables. An example of a general project may be to build a house, a building or bridge. Although this task may be complex, it is a visible, inflexible task and not subject to conformity. A software project, on the other hand,

- is invisible, for instance writing an operating system for a new computer;
- is complex, such as an Internet flight-booking system (or a flight control and scheduling system for an airline);
- should be able to conform, for instance amending a financial system to deal with different currencies (amending the federal tax system of the U.S. government);
- should be flexible, such as implementing a data warehouse for sales services and integrating various brands from various vendors placed on different databases and formatted differently, while allowing for the integration of information on new databases.

Schwalbe (2006) adds to the above characteristics by stating that a software project has a unique purpose; it is usually temporary; a software project is developed using progressive elaboration; a software project requires resources; a software project should have a primary customer or stakeholder, and a software project involves uncertainty.

The unique nature of SPM therefore contributes to the difficulties experienced with managing software projects and the likely failure of such projects.

3.3 CHANGING ENVIRONMENT OF SPM

SPM operates in a highly dynamic environment that involves temporary tasks and rapidly changing technology, and that requires coordination between various parties and organisations (Olson, 2004). In a cornerstone publication, *The Mythical Man-Month* by F.P. Brooks (1979), difficulties of managing large software development projects are identified and solutions proposed to solve these problems (Verner, Overmyer and McCain, 1999). The work describes pitfalls and fundamental problems and proposes suggestions for improvement of software project management. Over the past decade, computer technology expanded and management and control functions were automated and supported by software tools and techniques in an effort to support SPM and control (Chandrashekar, Mayfield and Samadzadeh, 1993).

Currently, the SPM environment is still changing due to business globalisation and information technology advances that support distributed and virtual teams and projects (Chen et al., 2003; Hughes and Cotterell, 2006; Callegari and Bastos, 2007). The increasing number of distributed projects involving software project collaborators from different locations, organisations and cultures, changes the SPM paradigm of a traditional project focusing on a single project executed at a specific location (Evaristo and Van Fenema, 1999; Jonsson, Novosel, Lillieskold and Eriksson, 2001; Olson, 2004; Smits and Pshigoda, 2007). Due to this distributed nature of software projects, high levels of collaboration are essential for successful project execution. Tracking of work processes, effective sharing of information and knowledge among collaborators, as well as proactive change management across time, space and technology are essential (Chen et al., 2003).

Literature reveals that several factors contribute to this changing environment of SPM:

- *The globalisation of the economy* has led to the geographical distribution of resources and investments in order to obtain better results (Zanoni and Audy, 2003; Dekkers and Forselius, 2007). A physically distributed environment

implies that users and development teams may be situated in different places, countries and possibly different cultures. The software development area has been foremost in this process in countries such as India and Ireland (Zanoni and Audy, 2003).

- The spreading of software development processes offshore or *outsourcing* is implemented in order to attain greater productivity, reduce cost and risk, and improve quality. The key issue for distributed software projects is coordination and collaboration (Chen et al., 2003).
- *Teams of people*, as representatives of the software project development effort, regularly collaborate with the software project and task leaders to plan, compose and monitor tasks (Cleetus, Cascaval and Matsuzaki, 1996; Gaeta and Ritrovato, 2002; Zanoni and Audy, 2003; Rose, Pedersen, Hosbond and Kraemmergaard, 2007). Such teams are commonly distributed over several dispersed geographical locations, and even several enterprises involving sub-contractors and sponsors. Distance may slow down interaction and communication.
- *E-business and Internet growth* stimulate and support the distribution of role-players involved in software development (Gaeta and Ritrovato, 2002). Although numerous new opportunities and possibilities are provided, complexity is increased and care must be taken to assure the quality of integrating the Internet with SPM processes.
- *Advances in technology*, such as distributed component technologies (Gaeta and Ritrovato, 2002) and parallel and distributed process architectures (Chen et al., 2005; McMichael and Lombardi, 2007), enable collaboration and concurrency, but also enhance complexity and maintenance problems.

The focus of SPM processes has clearly shifted away from the position that it held two decades ago. Consequently, tools for effective sharing of information and knowledge among project contributors, as well as efficient task scheduling, tracking and monitoring are needed. High levels of collaboration, task interdependence and

distribution have become essential across time, space and technology (Chen et al., 2003).

3.4 PROBLEMS WITH SOFTWARE PROJECTS

Several studies have been conducted to identify weaknesses and areas to improve during SPM. A number of surveys specifically investigated failed software projects (Verner, et al., 1999; Flower, 1996; The Standish Group, 1995, 2001, 2003 and 2005; Verner and Cerpa, 2005; Dekkers and Forselius, 2007), to list but a few on this topic.

According to the Standish group (2003), failure begets knowledge.

“If you begin with failure and learn to evaluate it, you also learn to succeed.”
(Olson, 2004)

Hughes and Cotterell (2006) identify the following aspects that, from the software project manager’s point of view, contribute to the failure of software projects: poor estimates and plans; lack of quality standards and measures; lack of guidance in decision making; lack of techniques to make progress visible; poor role definition. From the point of view of the team members the following are an indication of failure: inadequate specification of work; management ignorance of Information Communication Technology; lack of knowledge in application area; lack of standards; lack of current documentation; lack of communication.

A study of SPM practices in Australia reveals that fifty percent of software projects begin with unclear requirements, twenty percent has no life-cycle methodology, and risk assessment does not form part of the development process (Verner and Cerpa, 2005). Although the majority of managers identified risks at the start of the project, only half followed through during development.

Many of the problems identified with software project failure stem from poor communication and uncoordinated support to all team members. The question that begs to be asked is whether *standard SPM practices* are failing?

Practitioners have attempted to apply several Software Engineering (SE) principles to different SPM processes in order to address the existing shortcomings in the management of software projects (Lethbridge and Laganieri, 2001). Research investigations and software engineering textbooks compare and contrast different process models or life-cycle models for development (Kettunen and Laanti, 2004). Standard structured analysis and design methods are explored, while object-oriented approaches and extreme programming are used to overcome the aforementioned shortcomings (Lethbridge and Laganieri, 2001; Gelbard et al., 2002). Heuristics are explored (Purvis, McCray and Roberts, 2003) and visualisation is scrutinised to assist SPM processes (Hansen, 2006). Standard process definitions, process maturity assessment models and quality management systems such as the Unified Software Development Process and the Capability Maturity Model for Software Engineering are in use (Olson, 2004; Sonnekus and Labuschagne, 2004; Hughes and Cotterell, 2006). Standards such as ISO9001 have been formulated, and the compliance of the development process to these standards is tested. Different standards-based software project management approaches have been developed and are in use, such as PRINCE 2 and BS 6079 (Hughes and Cotterell, 2006). Yet, the results remain disappointing since many software projects still fail to comply with the triple constraints of time, cost and scope (Oghma: Open Source, 2003). These triple constraints refer to the fact that the failure of software projects can mostly be attributed to the fact that they are not delivered on *time* and do not meet the expectations of the client (*scope*), and as a result have *cost* implications. Sommerville and Rodden (1996) note that a large number of co-operation activities are unplanned, and that software engineers work in a flexible, autonomous fashion, which is not supported by the existing process models.

The traditional focus of SPM processes has shifted. Consequently, the size, complexity and strategic importance of information systems that are currently being developed require stringent measures to determine why projects fail. Since organisations continue to invest time and resources in strategically important software projects, the possibility of failure of the project should be minimised.

3.4.1 Minimising project failure

The 1995 Standish Group study found that the three major factors related to software project management *success* were user involvement, executive management support and a clear statement of requirements (The Standish Group, 1995). After their famous CHAOS report (1995), the Standish Group studied 13 522 projects in a follow-up survey, dubbed the EXTREME CHAOS report (2003). In the latter report, executive management support, user involvement, an experienced project manager and a clear statement of requirements topped the list of requirements for success. In 2005 the Standish Group reported that success rates increased to a third of all projects, but time overruns now measure 82 percent, whilst only 52 percent of required and specified functions and features were included in the final product. Their 2005 study determined that 18 percent of the surveyed projects failed, 53 percent did not meet the requirements (challenged) and only 29 percent succeeded.

Table 3.1 summarises the findings of the 2005 report by The Standish Group, describing in order of importance the factors that contribute most to the success of software projects:

Table 3.1 The Standish Group Report

1	User involvement
2	Executive management support
3	Clear business objectives
4	Optimising scope
5	Agile process

6	Experienced project manager
7	Financial management
8	Skilled resources
9	Formal methodology
10	Standard tools and infrastructure

Pinto and Slevin (1987) investigated the dual importance of strategy and tactics and after examining over 400 software projects, identified 10 critical success factors. They propose a framework involving conceptualisation, planning, execution and termination to illustrate these issues. To identify a solution to project failure is not an elementary process. SPM should be investigated as a whole, and each of its processes should be considered to fully understand the scope of SPM. In the following section SPM will be defined and its functions delineated.

3.5 SOFTWARE PROJECT MANAGEMENT FRAMEWORK

To discuss software project management, it is important to start off by defining a software project. Schwalbe (2006) defines a software project as a temporary endeavour undertaken to create a unique product, service or result. Software projects involve the use of hardware, software and networks to create a product, service or result. What exactly is then meant by software project management?

The IEEE defines SPM 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).

In an effort to visualise and contextualise the SPM operational environment, various frameworks and guidelines have been compiled. Hughes and Cotterell (2006) describe a framework of basic steps, Step Wise, in project planning. Step Wise covers the planning stages of a software project and not the monitoring and control

of it. The planning stages may, however, be adapted to support planning in any other methodology or standard of development. PRINCE 2 is a set of project management standards originally sponsored by the Central Computing and Telecommunications Agency (CCTA) for use in British government software projects. Schwalbe (2006) compiles a framework of key issues of importance during the SPM process. This framework combines the environment as well as processes or functions of software project management. Figure 3.1 illustrates these issues in a framework that contains the key elements in the field of SPM. The components of this framework are derived from Project Management Body of Knowledge (PMBOK) (2004) guidelines.

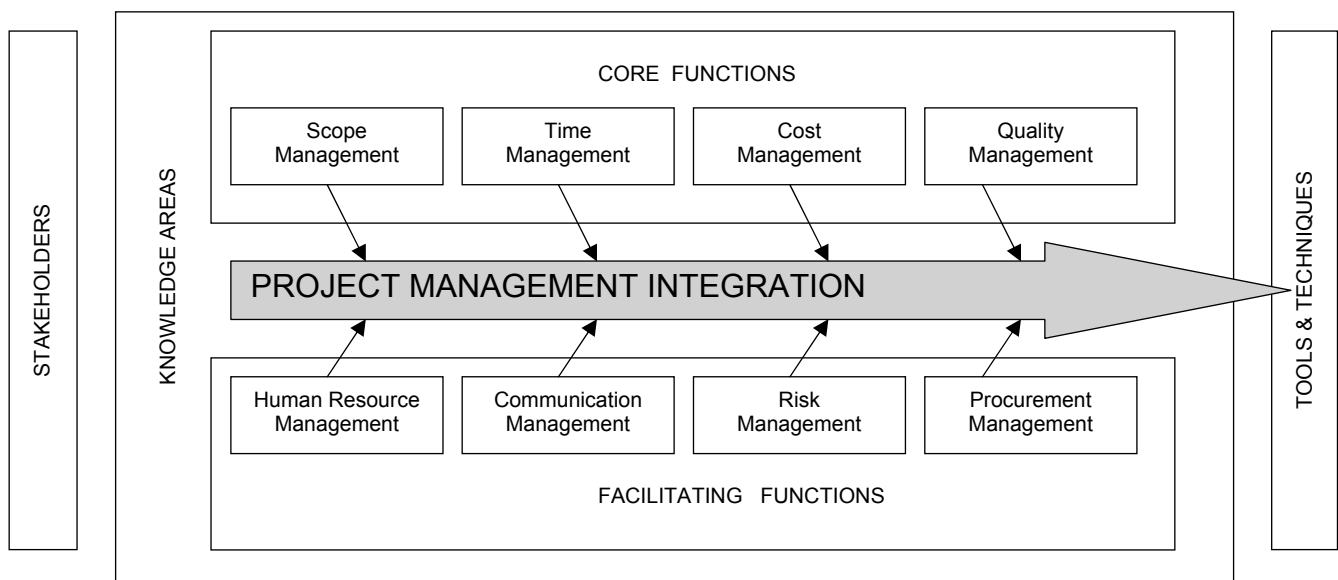


Figure 3.1 Software Project Management Framework (adapted from Schwalbe, 2006)

We distinguish between three key elements: software project stakeholders; software project management knowledge areas, namely core and facilitating functions; and software project management tools and techniques. The software project *stakeholders* and SPM *knowledge areas* comprise a working model. Software project management *tools and techniques* support these knowledge areas. This framework serves to explain the processes and refer to the SPM of a single project

in which the SPM manager allocates tasks and gives instructions to various role players.

The *software project stakeholders* are the people who are involved in the different project activities. They include the project sponsor, project team, support staff, customers, users, suppliers, as well as individuals with opposing views concerning the project. Good relationships, communication and coordination among all of these stakeholders are essential to ensure that the needs and expectations of stakeholders are understood and met. SPM *knowledge areas* include key functions concerned during the software project management process. The SPM knowledge areas consist of four core functions and four facilitating functions. The *core* functions – scope, time, cost and quality management – lead to specific project objectives and are supported by the *facilitating* functions. The *facilitating* functions represent the means to meet different objectives and include human resource management, communication, risk and procurement management. Project management *integration* is not regarded as separate function, but as supporting structure connecting all said functions with each other. Stretched across all these knowledge areas are the software project management *tools and techniques* (depicted on the right-hand side of the framework diagram in Figure 3.1). These are used to assist team members and software project managers to carry out their respective tasks.

3.6 CORE FUNCTIONS

Each of the core functions of the software project management development process (scope, time, cost and quality management) is discussed in the following section.

3.6.1 Scope management

According to Schwalbe (2006) the scope of a software project refers to all the work involved in creating the products of the software project and the processes used to create them. Being one of the first steps in the development phase, it is a difficult

but also important part of software project management, as problems or errors occurring during this phase will be perpetuated throughout the development process, and could be costly and time consuming to rectify at a later stage. The Standish Group's CHAOS study (1995) identified user involvement and clear statement of requirements as key factors associated with software project management failure.

To determine the exact scope of a software project, different approaches may be used. As the first phase of the Step Wise framework, Hughes and Cotterell (2006) propose the following steps:

- Identify objectives and measures of effectiveness in meeting them
- Establish a project authority
- Identify stakeholders
- Modify objectives in the light of stakeholder analysis
- Establish methods of communication with all parties.

Schwalbe (2006) and PMBOK (2004) identifies the following specific phases of software project scope management:

- *Scope planning* – the planning and refinement of project scope and the creation of a formal scope statement document that entails project assumptions, project constraints, a summary of all project deliverables, a description of the products involved in the project and a statement of what determines project success. The preliminary planning of one of the most important aspects of scope planning, namely the Work Breakdown Structure (WBS) is done at this stage.
- *Scope definition* – the division of major project deliverables into smaller and more manageable components. The deliverable of this phase is the WBS. This process is important to project success as it supports accurate time, cost and resource estimates and defines a baseline for project control and performance measurement. The WBS defines the total scope of the project by defining work packages.

- *Creating the Work Breakdown Structure (WBS)* – subdividing the project deliverables into smaller work packages.
- *Scope verification* – formal acceptance of the scope of the project by various key stakeholders.
- *Scope control* – managing all changes to the scope of the project. Aspects of importance to be recorded are not only scope changes, but also preventive action, corrective action, additions or deletions and lessons learned.

Various types of software are available to assist in project scope management, for instance word-processors for scope definition documents and spreadsheets for financial calculations. Automated software is also available for drawing the WBS.

3.6.2 Time management

Scope management entails determining the objectives and full functionality of the project. The next step is to determine the time and cost that the identified products and activities will entail, as well as the resources needed to develop the project. Time and cost aspects may overlap directly, thus influencing each other, and in practical terms they may or may not execute concurrently (Olson, 2004). As mentioned previously, many software projects fail with respect to scope, time and cost management. Managers often site schedule issues as one of the main reasons for conflict on projects.

Time management involves the processes required to measure timely completion of a project (Schwalbe, 2006). Time management involves not only the creation of an activity plan, but also the estimation of time required for each task and activity, resulting in the overall duration of the project. The output is the precedence network and critical path. This is also utilised throughout the duration of the entire project for scheduling, management and risk identification and management purposes. Time is

highly correlated with cost, and keeping a project on schedule is indeed a major challenge (Meredith and Mantel, 2002).

Activity planning forms the baseline for project and resource scheduling and supports a number of objectives, namely feasibility assessment, resource allocation, detailed costing, motivation and coordination of the project (Hughes and Cotterell, 2006).

The main processes of time management are the following (PMBOK, 2004, Schwalbe, 2006):

- *Activity definition* involves the identification of each task or activity that must be executed by stakeholders or project team members in order to produce the project deliverables. Thus it can also be seen as refining the scope of the project. An activity or task is an element of work with an expected duration, cost and resource usage. The work breakdown structure (WBS) forms the basis for this, and the initial WBS will be used to develop a more detailed WBS with supporting explanations as well as assumptions and constraints related to specific activities. This process may be performed using an activity-based approach or a product-based approach. In both cases a WBS and a product flow diagram (product-based approach) will be outputs.
- *Activity sequencing* involves indicating when each of the identified activities should occur. Activities in the detailed WBS will be reviewed referencing the detailed product descriptions, assumptions and constraints to determine the relationship between the activities. The output of this phase is the network diagram. An activity-on-node (precedence diagram) approach may be used. Network planning models were originally developed in the 1950s with the two best-known methods being CPM (Critical Path Method) or PERT (Program Evaluation Review Technique).
- *Activity resource estimation* includes the determination of how many and which resources will be needed to perform project activities.

- *Activity duration estimation* involves estimating the number of work periods to be completed in order to complete an individual activity. Duration estimation should not be confused with effort estimation. Effort estimation refers to the number of workdays or work hours required to complete a task, whereas duration refers to the time estimate and not the effort estimate. Effort and time are related, thus team members must record assumptions regarding both.
- *Schedule development* occurs when activity sequences and activity duration estimates, as well as resource requirements are combined to create the project schedule. The goal of this phase is to develop a realistic schedule that provides a basis for monitoring project progress for the time dimension of the project. Various tools and techniques are available for this process, including Gantt charts, CPM analysis, critical chain scheduling and PERT analysis.
- *Schedule control* refers to the control and management of changes to the initial schedule.

Software packages are available to support these steps. However, the concepts behind the software should be clearly understood, i.e. critical path or schedule baseline, in order to use these packages correctly.

Time management is often cited as one of the main sources of conflict with regard to projects (Olson, 2004). Closely linked to activities and the scheduling of activities, is the estimation of the cost of each activity, or then the cost of different components of the project.

3.6.3 Cost management

Cost estimation and monitoring occurs throughout the entire life cycle of the project. It has also been identified as one of the major causes of project failure, together with time and user requirements. In the 1995 CHAOS project report, average cost overruns of software projects tested were cited as 189 percent of the original

estimates. The 2001 CHAOS report reflected an improvement from 189 percent to 145 percent, but this means that the projects still exceeded their associated budgets substantially.

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 is an important part of the initial assessment of whether the project will be feasible or not. An economic assessment of a proposed information system will be made by comparing the expected costs of development and operation of the system. Thus, basic project cost estimates throughout the development of the project will involve the following:

- *Cost estimating* refers to the process of developing an estimate of the costs of all actions, resources and procedures needed to complete the project. Cost estimation should include development costs and operational costs where cost estimation is performed during different stages of product development. Basic accounting and financial principles are used for the initial cost-benefit analysis plan included in the feasibility study, such as cost-benefit analysis, cash flow analysis, calculation of internal rate of return, net profit, payback period, return on investment and net present value. Various estimation tools and techniques exist, namely analogous estimating, bottom-up estimating, parametric modelling (such as amongst others COCOMO) and computerised models.
- *Cost budgeting* involves using the project cost estimate and allocating this to individual work items. The WBS, as well as the project schedule is used to allocate costs over time. A cost baseline is a time-phased budget that can be used to measure and monitor cost by managers or project leaders.
- *Cost control* includes monitoring cost performance, reviewing changes and notifying stakeholders and team members of changes related to cost.

Cost control cannot be done without a clear identification of resources to be used. Various computerised software packages exist to support this action.

3.6.4 Quality management

The Project Management Body of Knowledge (PMBOK) (2004) defines project quality management as the processes required to ensure that the project will satisfy the needs for which it was undertaken (Schwalbe, 2006). 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. Quality management does not only include 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 PMBOK (2004) are the following:

- *Quality planning* determines which quality standards are relevant to the specific project under consideration and decides how these standards will be met. Hughes and Cotterell (2006) state that this process should identify variables that have a direct influence on the outcome of the project. Thus, aspects such as functionality, system outputs, performance, reliability and the quality policy should be included in the quality management plan.
- *Quality assurance* involves evaluating overall performance regularly to ensure conformance to the set standards. Quality audits or reviews support this function throughout the project. Tools to be utilised include quality audits, templates specifying required documentation, quality assurance procedures, problem-reporting procedures, quality assurance metrics and quality assurance check list forms (Hughes and Cotterell, 2006).
- *Quality control* monitors the activities and end results of the project to ensure compliance to the standards utilising various available tools and techniques. The quality control process mainly consists of decisions to determine if the products or services produced will be accepted or rejected (if not accepted rework is specified on the items), and process adjustments to correct or prevent further quality problems (Hughes and Cotterell, 2006). Various tools and techniques may be utilised in this phase.

However, quality management should not be considered as a separate development phase. It should be integrated into all phases and all processes during software project management.

Process as well as product quality measures should be implemented. In an effort to give structure and uniformity to this process, several standards and measures have been developed over the past number of years. These standards and measures will be discussed in the following section.

3.6.4.1 Existing Standards

Software development is a fast growing industry and the lack of standards holds significant consequences for society and the economy (Nienaber and Barnard, 2005). In an attempt to solve this problem, various national and international standards bodies have proceeded to set standards for this area of development. The standards listed below are of importance:

- 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 developed further and the PMBOK Standards were published in 1987, whilst the Guide to PMBOK was published in 1996. PMI furthermore published the Organizational Project Management Maturity Standard in 2003, which will be revised and published in December 2008. Currently the PMI is working on the OPM3 as the global standard for organisational project management (Project Management Institute, 2004).
- The ISO standard 9126 was published in 1991 to address the problem of defining software quality (Hughes and Cotterell, 2006). 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 to the specific quality metric.

- Standards set in the UK are among others PRINCE (set by the Central Computer and Telecommunications Agency) and BS 6079, which both apply to any type of project. The British Standards Institution (BSI) set the BS 6079:1996 standard identical to the international standard ISO 9000:2000, followed by the 2001 and 2004 standards respectively.
- The primary objective of standards such as the ISO 9000 series is to ensure that a monitoring and control system to check quality is in place. The ISO 9000 series targets the fundamental features of a quality management system (QMS), thus quality in general terms, and not specifically quality in software project management. ISO 9001 describes the creation of products and the provision of services, while ISO 9004 targets process management (Hughes and Cotterell, 2006). ISO 10006:2003 targets a quality management system and provides guidelines for quality management for projects specifically.
- The capability maturity model (CMM) was developed at the Software Engineering Institute in the United States (McBride, Henderson-Sellers and Zowghi, 2004; Schwalbe, 2006). This model defines different stages of process maturity, implying sophistication and quality of production practices at which an organisation may be placed. 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 (Hass, Johansen and Pries-Heje, 1998).

3.6.4.2 Product quality measures

Measurement of quality usually concerns intangible, invisible factors. Hughes and Cotterell (2006) define practical software quality measures such as reliability that might measure availability, mean time between failures, failure on demand and support activities. Other practical measures include maintainability and extendibility. Various factors that enhance quality have been identified over the years in an attempt to improve quality measures, but lack of conformity of definitions and terms

pose a problem. Thus, the difference between measures, techniques and approaches is not outlined clearly and the approaches overlap.

Hughes and Cotterell (2006) cite the following measures to enhance quality:

- Increasing visibility of the development process involves utilising egoless 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 the 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 (Hughes and Cotterell, 2006). Different approaches to quality control are also utilised. Mehandjiev, Layzell, Brereton, Lewis, Mannion and Coallier (2002) state that a goal-driven approach is more appropriate to handle adaptability and productivity requirements, whereas Szejko (2002) promotes requirements-driven quality control.

3.7 FACILITATING FUNCTIONS

The facilitating functions of the SPM framework represent the means through which different objectives are to be met. Human resource management, communication management, risk management and procurement management form part of these functions. The following section will focus on these facilitating functions.

3.7.1 Human resource management

Human resource management involves all processes required to effectively utilise all resources involved in a project (Schwalbe, 2006). A resource may be seen as any item or person required for the execution of a project. Human resource management therefore concerns all project stakeholders involved in developing the project. Hughes and Cotterell (2006) identify seven categories of resources to be

managed for a project, namely labour, equipment, materials, space, services, time and money.

The main focus of human resource management is to allocate these resources to activities, and to create a work schedule based on the activity plan. Activities should be scheduled to minimise variations in resource levels during the project and resources will then be allocated to competing activities.

Schwalbe (2006) identifies four phases to achieve the above, namely human resource planning, acquiring the project team, developing the project team and managing the project team.

- *Human resource planning* involves identifying, assigning and documenting project role players, their roles and relationships. An organisational chart may be an output of this phase.
- *Acquiring the project team* involves appointing or identifying and assigning the appropriate personnel to the project activities.
- *Developing the project team* includes improving team skills to enhance project performance. Various theories exist for managing people and teams, but these aspects fall outside the scope of this document.
- *Managing the project team* also includes the monitoring and controlling of resources throughout the project. Progress must be monitored and software for visualising progress can support the project team.

Project management software and general software are available to support human resource management. Examples are:

- Spreadsheets
- Project organisational charts
- Responsibility assignment matrices
- Resource histograms

However, project resource management involves much more than using software to facilitate organisational planning and assigning resources. As this phase involves

people, psychosocial issues that affect how people work and how effective they work, must be recognised as influential issues during this phase.

3.7.2 Communications management

The 1995 Standish Group study found that the three major factors related to software project success were user involvement, executive management support and a clear statement of requirements (Krupansky, 2003). All of these project success factors depend on good communication and coordination skills among the stakeholders. Poor, ineffective or untimely communication, contradictions, omissions, failure to notify all of meetings and decisions, and failure to store information are often cited as reasons for projects failing or running over time. Traditional reporting tools use a simple passive reporting mechanism, which does not provide sufficient reporting support to a collaborative distributed system (Chen et al., 2003).

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 needed. Schwalbe (2006) identifies four distinct functions associated with communications management:

- Communications planning
- Information distribution
- Performance reporting
- Stakeholder management

The *communications planning* function determines the *who*, *when* and *how* of the project. It therefore facilitates a collaborative working environment that determines the information and communication needs of the stakeholders. The *information distribution* function entails disseminating information to keep all the stakeholders informed. *Performance reporting* alludes to the generation of reports such as status,

progress and forecasting reports, while the *stakeholder management* function involves managing stakeholders, approving change requests and updating reports.

Communication can, however, be enhanced and supported by the use of a common repository. A paper-based repository has several disadvantages, such as retrieval delays, lost documentation and error proneness, but most of all, it may result in insufficient project documentation in the distributed environment. Another common problem with regard to communication is that many project processes, contexts, rationales or artefacts may not be captured at all. An electronic repository promises to overcome some of these disadvantages.

3.7.3 Risk management

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.

Schwalbe (2006) describes project risk management as the art and science of identifying, analysing and responding to risks throughout the life cycle of the project. The Webster's Dictionary defines risk as hazard, peril or exposure to loss or injury, whereas the PMBOK defines project risk management as the systematic process of identifying, analysing and responding to project risk. The objective of risk management is to minimise or avoid the adverse effects of unforeseen events. Many projects do not follow a formal risk management plan, which leads to a state of perpetual crisis. According to Marchewka (2003) the reasons for this may be that

- benefits of risk management are not clearly understood by the project leader;
- adequate time is not provided for risk management, and
- risk is not identified and assessed using a standardised approach.

Project management and planning is based on the understanding of various role-players of the current situation, the information available and the assumptions to be made. But since environments may change dynamically, events may not proceed

according to plan and various degrees of uncertainties exist that cannot be predicted with total accuracy. To ensure eventual success, those unexpected events must be addressed and managed throughout the life cycle of the project to ensure that project risk is minimised.

Various models or frameworks exist to ameliorate the risk associated with software project development. Schwalbe (2006) suggests the following risk management processes, namely risk management planning, risk identification, qualitative and quantitative risk analysis, risk response planning, and risk monitor and control. Based on Boehm's (1991) model, software risk management entails *risk assessment* and *risk control* (Figure 3.2).

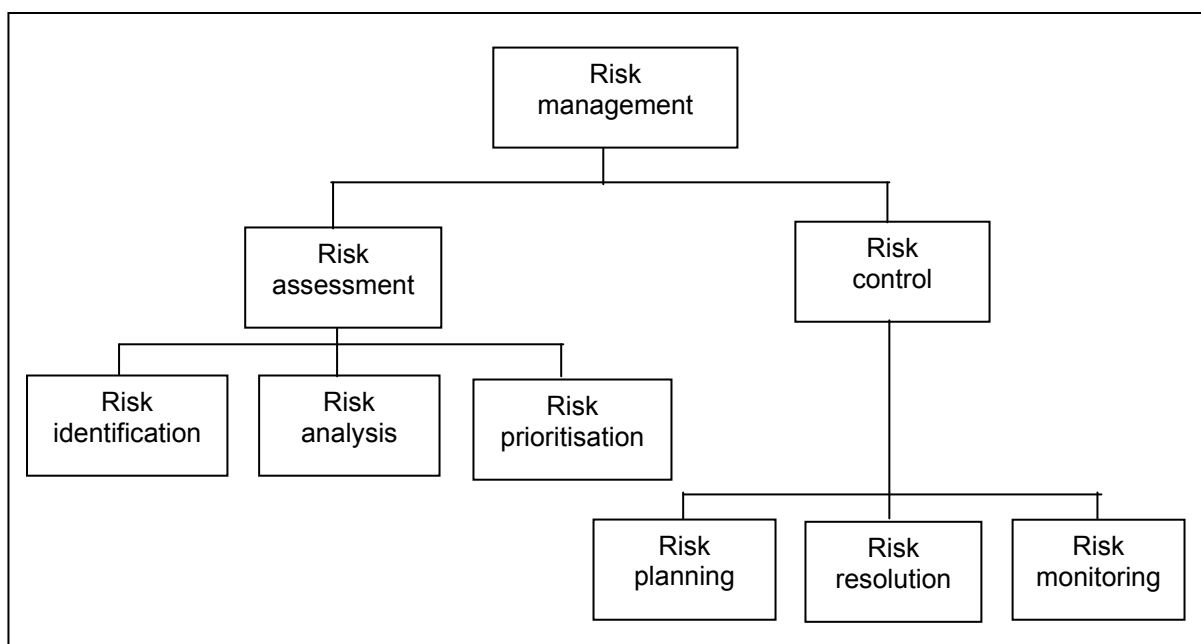


Figure 3.2 Software risk management (adapted from Boehm, 1991)

The first step deals with the assessment of the risks by means of stakeholder interaction with the SPM interface and as such, is dependent on the project manager's input on the potential risks in the software project. Potential risks are identified by using checklists, assumptions and decomposition, and analysed by means of performance models, cost models, network and decision analysis. Risk

prioritisation involves risk exposure, risk leverage and compound-risk reduction. The second step deals with the control of these risks, and comprises risk planning, risk resolution and risk monitoring. Risk planning entails the drawing up of contingency plans to counteract each identified risk, risk avoidance strategies, risk transfer and risk reduction plans. Risk resolution entails the elimination or resolution of risks by using prototypes, simulations and benchmarks. Risk monitoring involves the tracking of the project's progress towards resolving its risk items. For effective risk management, risks should be identified, discussed and monitored as early as possible (Phillips, 1998).

3.7.4 Procurement management

During the process of project development it may be necessary to procure products, goods or items that are not available within an organisation (Marchewka, 2003). Procurement refers to the process of acquiring goods or services from an outside source. Procurement management refers to a set of procedures for acquiring these products, expediting external work and ensuring a 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 and Cotterell, 2006). The BS 6079 document describes these aspects in more detail (Hughes and Cotterell, 2006).

Although the term 'procurement' is widely used, information technology professionals also refer to 'outsourcing', while other private companies prefer the term 'purchasing'. As the outsourcing of important information technology functions is increasing at a tremendous rate, this area should be considered very important in relation to the success of a project. According to Schwalbe (2006), organisations outsource to reduce costs, focus on their core business, access skills and technologies, provide flexibility, and increase accountability.

Project procurement management consists of six main processes: procurement purchasing and acquisition; planning contracting; requesting seller responses; selecting sellers; administering the contract; and closing the contract (Schwalbe, 2006).

Procurement purchasing and acquisition involves decisions as to what to purchase, when and how. The possibility to outsource must be considered. Outputs of this process are the procurement management plan, contract statement-of-work, and make-or-buy decisions. Existing tools and techniques include performance make-or-buy analysis, and internal as well as external expert consultation. The type of contract for the above-mentioned products should be determined. Various options exist, such as a fixed-price or lump-sum contracting, cost reimbursable contracting, time and material contracts, and unit price contracts.

Planning contracting refers to describing requirements for the products or services desired from the procurement and identifying potential sources or sellers. This phase involves writing procurement documents such as a Request for Proposal (RFP), and developing source selection evaluation criteria for the entire organisation.

Requesting seller responses entails obtaining quotes, information, bids and offers. The main outputs of this process include a qualified sellers list, procurement package and proposals.

Selecting sellers involves evaluating potential sellers and negotiating the contract.

Administering the contract entails managing the relationship with the selected seller. Outputs of this process are contract documentation, requested changes, corrective actions and updates.

Closing the contract involves the settlement and completion of each contract.

3.8 CONCLUSION

In this chapter the unique characteristics and changing environment of software project management were explored. The diverse nature of SPM, as well as the wide range of business areas and technologies involved makes it especially challenging to manage. The dynamically changing environment adds to the complexity of these systems and results in higher levels of interconnectivity, higher levels of sharing data and knowledge, and higher levels of task tracking and monitoring. These issues should be optimally supported by SPM processes to enable project managers to concentrate on crucial issues and strive for lower failure and higher success rate in software projects.

The SPM processes have been delineated in order to get a clear picture of the entire SPM process in an effort to determine the aspects that drive project failure. Based on the literature survey conducted in this chapter, the researcher defines the phases of each of the eight knowledge areas of SPM, as depicted in Table 3.2:

Table 3.2 Phases of the knowledge areas of SPM

Scope Management	Time Management	Cost Management	Quality Management	HR Management	Communication Management	Risk Management	Procurement Management
Initiation	Activity definition				Identification and planning	Risk identification	Procurement planning
Planning	Activity sequencing; Duration estimation	Resource planning	Planning	Organisational planning	Team support	Risk analysis and prioritisation	Solicitation planning
Definition	Time schedule development	Cost estimation	Assurance	Team development and staff acquisition	Information distribution	Risk management planning	Solicitation and source selection
Verification	Time schedule control	Cost budgeting	Control	Management: monitor and control	Performance reporting	Monitor	Contract administration
Change		Monitor;			Administra-	Resolution	Contract

Scope Management	Time Management	Cost Management	Quality Management	HR Management	Communication Management	Risk Management	Procurement Management
control		control			tive closure		closure

These phases will form the basis of the SPM model that will be introduced in chapter 6.

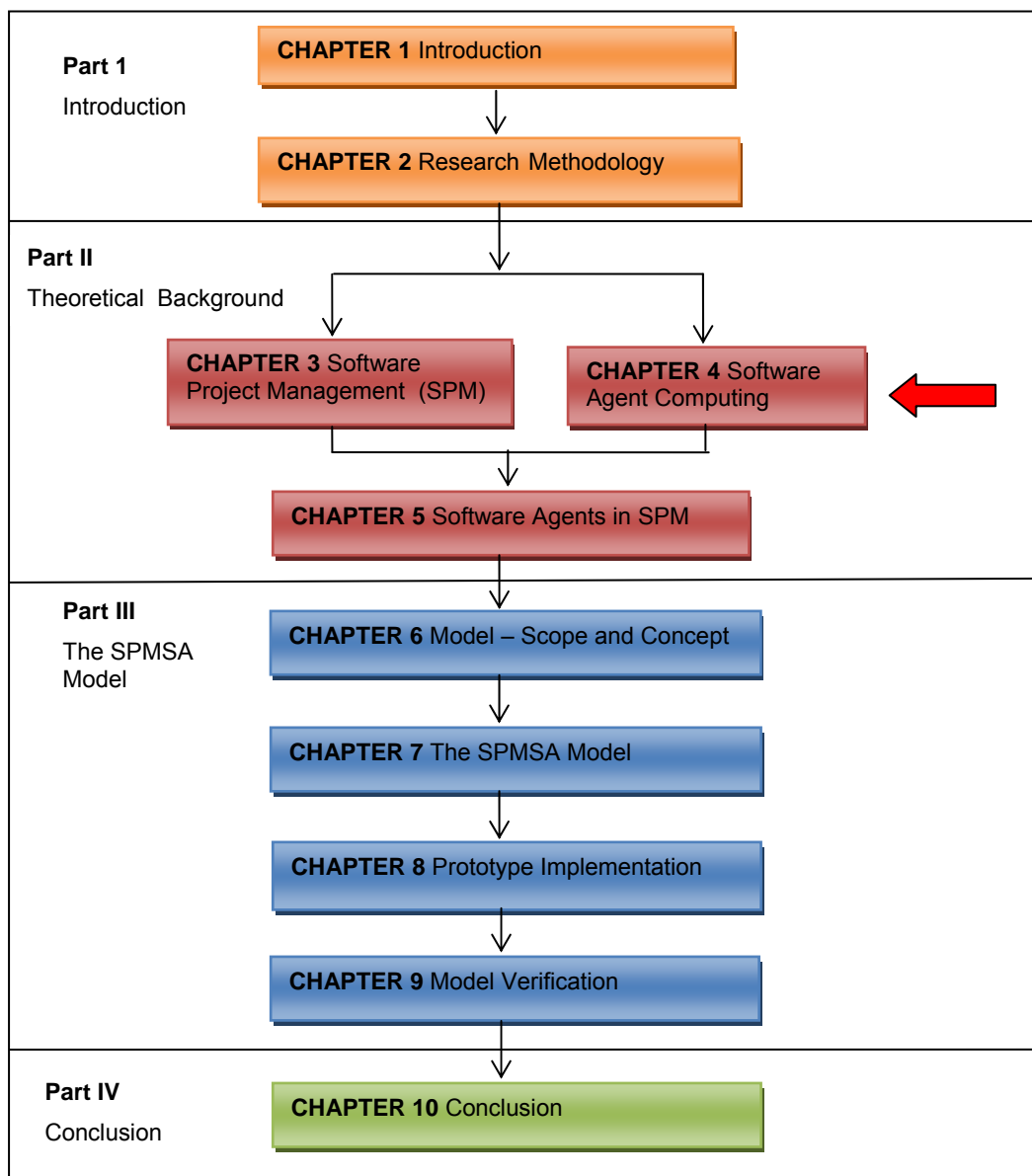
According to research studies done software projects still regularly fail to comply with requirements and expectations (The Standish Group, 2005). Organisations continue to invest time and resources in strategically important software projects, thus one would expect the possibility of project failure to be minimised. Failure in many areas of SPM indicates *shortcomings in existing standard practices* (Schwalbe, 2006; Hughes and Cotterell, 2006) while research shows that there is an *urgent need for successful SPM practices* (Marchewka, 2003). It seems essential to address the unique and changing environment of SPM if any success is to be obtained.

This study focuses on enhancing existing SPM practices. Despite the existence of various SPM application packages, as well as enterprise management packages, none of them successfully address the dynamically changing and unique environment of SPM (Schwalbe, 2006).

In Chapter 4 the use of software agents will be investigated as a potential tool to support software project management. We specifically concern ourselves with the question whether software agents can be used to support and enhance SPM in a distributed environment and in this way address the limitations of current practices.

CHAPTER 4

4 SOFTWARE AGENT COMPUTING



4.1 INTRODUCTION

In the previous chapter the changing environment of SPM was explored to identify problems experienced during software project management processes. The changing environment and unique nature of SPM are seen as factors that contribute to the failure of software projects. The researcher investigated the SPM environment and accordingly compiled phases of the eight knowledge areas of SPM, which will form the basis of the SPM model that will be introduced in chapter 6.

The purpose of this chapter is to explore software agent computing to determine if it can be utilised to support software project management processes. To orientate the reader with regard to the background of agents, and more specifically mobile software agents, concepts and identifying features of agent technology and mobile agents are discussed. The trends and drivers for agent technology are presented, as well as a brief history of mobile agents. The goal of this chapter is to consider software agent technology from this viewpoint to determine whether it can support SPM and, as a result, minimise project failures.

4.2 AGENT TECHNOLOGY

Computing evolved through different metaphors in its lifetime. Charles Babbage in the nineteenth century saw computation as calculation or operations on numbers. With the advent of widespread digital storage and manipulation of non-numerical information, computation was replaced by the term information processing. The development of the Internet and the World Wide Web has necessitated a new metaphor, namely computation as interaction (Luck, McBurney, Shehory, Willmott, and Agentlink Community, 2005). This metaphor identifies computing as actions taking place by and through communication between computational entities. Monolithic architectures are replaced by distributed systems and autonomous components (Rigaud and Guarnieri, 2002).

New open and dynamic environments in which heterogeneous systems must interact span organisational boundaries. The fact that this type of systems should operate effectively within rapidly changing circumstances and with dramatically increasing quantities of available information, suggests the need for new computing models and paradigms (Luck et al., 2005).

Agents can be viewed as a new metaphor of computing. In a publication by Luck et al., (2005), two main stages of adoption are identified through which new technologies evolve: an installation period of exploration and development; and a deployment period concentrating on the use of such new technology. Agent-based computing is considered disruptive and results in a re-evaluation of the nature of computing. According to Franklin and Graesser (1996) a software agent is an autonomous system that, when situated within an environment, forms part of the said environment. The software agent can sense the environment and act on it over a period of time. However, various descriptions, views and definitions of agents exist.

Software engineering explores the complexity of computer systems and stipulates interaction as one of the most important components of complex software. Software architectures that contain more than one dynamically interacting component, each with its own thread of control and engaged in complex coordinated protocols, are exponentially more complex to design and control than single function systems. Agents are seen as a paradigm for ubiquitous, interconnected computer systems (Wooldridge, 2002). Agents can also be viewed as a tool to understand *human societies*. Doran, Franklin, Jennings and Norman (1997) used multi-agents to simulate an ancient society in order to shed light on some social processes. This implies some sort of intelligence. Agents have been viewed as a type of *distributed system*, or subclass of distributed system (Ben-Ari, 1990). The agent system however exhibits autonomous action, which differ from concurrent systems. Since agents may or may not exhibit intelligence, the area of *artificial intelligence* is also related to agent computing. According to Wooldridge (2002), economics theory and

social science are also related to the concept of agents. Luck et al. (2005) consider agents as a design metaphor, source of technology, as well as a method for simulation. It is clear that agent technology spans a diverse area and can be seen as related to different disciplines; yet it exhibits characteristics of its own.

4.2.1 Emerging trends as drivers for agent technology

Agent computing provides explicit benefits for these new open and dynamic environments. The AgentLink group published a 'Roadmap to agent-based computing' (2005) and cited the following trends as drivers for agent computing: the semantic web; web services and service-oriented computing; peer-to-peer-computing; grid computing; ambient intelligence and self-*systems and autonomic computing.

4.2.1.1 The semantic web

The semantic web is based on the concept that data on the web can be defined and linked in such a way that it can be used by machines for automatic processing and integrating of data across different applications (Berners-Lee, Hendler and Lassila, 2001). The key to enabling the sharing and processing of data on the web is by augmenting web pages with descriptions of their content in such a way that it is possible for components to reason automatically about that content. Agent-based systems can therefore be built on top of the semantic web and thus exploit its value. Luck and d'Inverno (2004) suggest that the semantic web demands effort and involvement from the field of agent-based computing and that the two fields are intimately connected.

4.2.1.2 Web services and service-oriented computing

Web services provide a standard means of interoperating between software applications on different platforms (Luck et al., 2005). Standards for a wide range of interoperability issues such as basic messaging, security, architecture and service discovery (produced by bodies such as W3C and OASIS) provide a framework for the deployment of component services that are accessible and use HTTP and XML interfaces.

Web service standards serve as a potential convergence point for diverse technology efforts such as eBusiness frameworks (ebXML, RosettaNet, etc.). Thus, web services provide a ready-made infrastructure for supporting agent-based systems. An agent-oriented view of web services is gaining exposure, as provider and consumer web services can be seen as a form of agent-based system (Booth, 2004).

4.2.1.3 Peer-to-peer computing

Another technological development that serves to support agent-based systems is peer-to-peer (P2P) computing. A wide range of infrastructures, technologies and applications constitute P2P computing. These applications are designed to create networked applications where the deployed system (or every node) is equivalent and application functionality is created by arbitrary interconnection between the peers. A range of agent-like characteristics, such as self-organisation, behaviour and negotiation, are displayed by P2P systems.

4.2.1.4 Grid computing

Kephart and Chess (2004) state that grid computing has recently gained interest as a high-performance computing infrastructure for supporting large-scale distributed scientific endeavours. The grid provides a computing infrastructure for supporting large-scale information handling, knowledge management and service provision.

This infrastructure is abstracted into several layers, which may include a data layer for resource allocation, scheduling and allocation; an information layer for the handling of information; and a knowledge layer. This infrastructure enables the integrated collaborative use of computers, networks, databases and scientific instruments of various organisations.

4.2.1.5 Ambient intelligence

The vision of ambient intelligence relies on ubiquitous computing, ubiquitous communication and intelligent user interfaces. Ambient intelligence was identified by the European Commission as a challenge for research and development in

information technology (Luck et al., 2005). This refers to an environment of potentially thousands of embedded and mobile devices or software components, interacting to support user goals and activity. Key features are autonomy, distribution, adoption and responsiveness, and in this sense they share the same features as agent-based systems.

4.2.1.6 Self-*Systems and Autonomic Computing

Luck et al. (2005) identify self-*systems (pronounced “self-star”) as a computational goal since the work of Charles Babbage. Although a general definition of these systems is still emerging, it includes properties such as self-awareness, self-organisation, self-configuration, self-management, self-diagnosis, self-correction and self-repair. Computational self-*systems provide an application domain for research and development of agent technologies, as many self*-systems may be viewed as involving interactions between autonomous entities and components.

Autonomic computing, first proposed by IBM (Kephart and Chess, 2003), is an approach to self-managed computing systems, with a minimum of human interference. Kotz and Gray (1999) add trends affecting Internet technology and activity, such as bandwidth, mobile devices, mobile users, intranets and aspects such as information overload, customisation and proxies.

4.3 WHAT IS A SOFTWARE AGENT?

A number of different descriptions and definitions of agents can be found in literature. Franklin and Graesser (1996) describe an autonomous agent as “a system situated within a part of an environment that senses that environment and acts on it, over time, in pursuit of its own agenda and so as to effect what it senses in the future”. They divide agents into taxonomies of biological, robotical and computational agents, and further subdivide computational agents into artificial life agents and software agents. The focus of this thesis is primarily on *software agents*.

The usefulness and viability of software agents have been debated since the mid-nineties. Various definitions for software agents exist but unfortunately those that are formulated are vague. In general, end-users see software agents as programs that assist people and act on their behalf by allowing people to designate work to them (Lange and Oshima, 1998; Grimley and Monroe, 1999). Jennings and Wooldridge (1998) define a software agent as an autonomous system, capable of flexible autonomous action in order to meet its design objectives. Another definition of a software agent states that it is a computer program that is capable of autonomous (or at least semi-autonomous) actions in pursuit of a specific goal (Franklin and Graesser, 1996). In the AgentLink publication, 2005, software agents are viewed as a computer system that is capable of flexible autonomous action in dynamic, unpredictable, typically multi-agent domains. 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 (Wooldridge, 2002).

The software agent is able to perceive the environment and act on it over a period of time. This corresponds to a widely accepted notion of agency which regards an agent as an autonomous software system acting in a continuous Perceive-Reason-Act cycle (as illustrated in Figure 4.1) in order to achieve a goal (Lind, 2001; Luck et al., 2005; Schoeman, 2005). The software agent *perceives* the environment by receiving and processing certain stimuli. It *reasons* by combining newly acquired information with its own existing knowledge and goals. It *acts* by selecting and executing one of the possible actions.

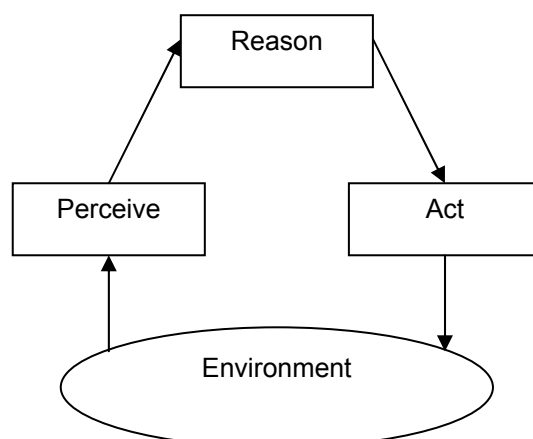


Figure 4.1 The Perceive-Reason-Act cycle

4.3.1 Classes of software agents

Software agents can be grouped according to specific characteristics and into different software agent classes. Literature does not agree on the different types or classes of software agents. For example, Krupansky (2003) distinguishes between *ten* different types of software agents, while the Oghma Open Source (2003) web site identifies sixteen different types of software agents. Because 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 thesis, we distinguish between two simple classes of software agents, namely *stationary agents* and *mobile agents*.

Agents that do not move are called *stationary agents*. A stationary agent executes only on the system on which it begins execution; it may typically use a communication system such as remote procedure calling. An agent is implemented as a code component and a state component. Furthermore, agents need an agent execution environment at both the client and host computers. This execution environment allows agent interaction, information exchange, agent mobility and agent security (Lingau, Drobniak and Domel, 1995). The execution environment will be discussed in Chapter 5. *Mobile software agents* on the other hand, are programs that can migrate from host to host in a network, at times and at places of their own choosing (Kotz and Gray, 1999). The state of the running program is saved,

transported to the new host, and restored. For the purpose of this thesis our focus is on *mobile* agents as discussed in detail in Section 4.4. Mobile agents are able to transport themselves from one machine to another. This ability of mobile agents are extremely suitable to a SPM environment as it will support distribution between teams and processes.

Agents in both these classes might or might not display any or a combination of the following characteristics: a user interface, autonomy, intelligence, adaptivity, flexibility and collaborative properties.

- Whether or not an agent has a *user interface* depends on whether it collaborates with humans, other agents or hosts. User interfaces are commonly found only where agents interact with humans.
- *Autonomy* in agents implies, as stated earlier, that the software agent has the ability to perform its tasks with minimum supervision.
- According to Wooldridge (2002), *intelligence* implies the inclusion of at least three distinct properties, namely *reactivity*, *proactiveness* and *social ability*. The *intelligent dimension* represents the agents' capability to express preference, beliefs and emotions, and their ability to complete a task by reasoning, planning and learning.
 - *Reactivity* refers to an agent's ability to perceive its environment and respond timeously 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. Different definitions attempt to define the collaborative nature of software agents.
- *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 customise 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 and Jiang, 2000).
- 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 among agents (Croft, 1997). It is expected of most agents to have a strong collaborative nature without necessarily implying other intelligence properties.

Agents may have additional abilities such as mobility, being rational, the ability to learn, and many other (Nwana and Ndumu, 1996a; Franklin and Graesser, 1996; Luck et al., 2005). Mentalistic attitudes, such as knowledge, belief, intention and obligation may also be added to the agent (Wooldridge, Jennings and Kinny, 2000; Dale, 1997; Luck et al., 2005). Intelligence in the form of reasoning and understanding can be added to agents to determine their behaviour and focus their reaction in given situations. Thus the spectrum of agents may vary from agents with no intelligence but performing useful tasks, to agents with intelligence (Dale, 1997).

4.4 MOBILE SOFTWARE AGENTS

Mobile agents are agents that are capable of transmitting themselves, their program and their state, across a computer network (Wooldridge, 2002). The state of the agent refers to its attribute values that direct it in determining what to do when it resumes execution at its next destination (Lange and Oshima, 1998). The mobile agent is able to transport its state and code when traversing a network to another execution environment in the network, and then resume execution. Code implies the class code necessary for the agent to execute.

Mobile agents conform to the criteria for *weak* agency, namely autonomy, social ability, reactivity and pro-activity (see Table 4.1). These are the criteria that enable agents to succeed in assisting humans (Jennings and Wooldridge, 1998; Lange and Oshima, 1998; Luck et al., 2005).

Table 4.1 Agent Properties

Property	Meaning
Autonomy	Operates independently without direct programmer or user intervention.
Social ability	Communicates with the local environment, other agents or user.
Reactivity (external)	Responds in timely fashion to changes in its environment.
Reactivity (internal)	Changes its behaviour based on experience.
Pro-activity	Exhibits goal-directed behaviour.

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 to interact with that object. During execution, the agent transfers information to the host and may receive information from the host (social ability). The agent will then decide, based on this exchange of evidence (reactivity), whether to terminate, become a resident agent, or repeat the migration process (pro-activity).

4.4.1 History of mobile agents

Mobile agent technology has been under development since about 1994 (Green et al., 1997). The concept of a mobile agent has grown from advances in distributed systems research. Initially mobile agents were intended to improve on remote procedure calls (RPCs) as a way for processes to communicate over a network (Wooldridge, 2002).

During the first wave of the computer industry, from 1970 to 1980, large proprietary mainframes were commonly used and procedural languages such as Cobol, RPL, Fortran and C were used. The mid-nineties constitute the second generation of the computer industry. With the advent of personal computers that became cheaper but

more powerful, PCs connected to networks and servers were commonplace and new programming paradigms such as object-oriented languages became popular: C++, Java, as well as visual programming such as Delphi and Visual Basic. We believe that we are currently experiencing the third generation, dispersed by industry standards, the Internet and mobile computing.

In a white paper describing General Magic's Telescript technology, Jim White (1995) coined the term 'mobile agent' to describe a procedure that travels with its data to the host computer, to be executed at the host. Typescripts was followed by research systems such as Tacoma in 1995, which offers operating system support for mobile agents, and Agent Tcl (currently known as D'Agents) in 1996.

Mobile agent systems are required to execute on a variety of hardware platforms, as well as to be implemented in code that does not require recompilation after migration. As a result, most existing mobile agent platforms are based on either scripting languages or Java. Java is currently the language used most commonly to program mobile agents (Green et al., 1997; Wooldridge, 2002; Luck et al., 2005). Well-known Java-based applications include Aglets, Voyager and Odyssey. IBM's Tokyo research laboratory started development of one of the first industrial mobile agent systems, Aglets, in 1995, while ObjectSpace's Voyager started in mid-1996, Mitsubishi's Concordia in 1997 and General Magic's Odyssey in 1997.

Although various mobile agent applications are developed over a large application area, agent implementation is still in the early stages of adoption. Application areas range from AI, telecommunication, distributed computing, intelligent user interfaces, e-commerce, power system management and air traffic control, to information retrieval management, smart databases, digital libraries and many more (Wooldridge et al., 2000; Luck et al., 2005).

In an effort to enhance acceptance of agent technology, two main sets of standards have been compiled. They are the Mobile Agent System Interoperability Facility (MASIF) from the Object Management Group (OMG) in 1998 (Milojicic, Breugst,

Busse, Campbell, Covaci, Friedman, Kosaka, Lange, Ono, Oshima, Tham, Virdhagriswaran and White, 1998) and the Foundation for Intelligent Physical Agents (FIPA) standard from the Agent Management Support for Mobility Specification (*FIPA00087 – FIPA 98 Part II Version 1.0: Agent Management Support for Mobility Specification*) in 1998.

4.4.1.1 The MASIF standard

The Object Management Group's (OMG) MASIF standard addresses interfaces between mobile agent systems. It also presents standards for cross-system communication and administration (Kotz and Gray, 1999). MASIF uses two primary interfaces for this purpose, namely the MAFAgentSystem and MAFFinder interface.

The above two interfaces address the following concerns:

- A standard to manage agents, including operations such as creation, suspension, resumption and termination
- A common mobility infrastructure for agent communication and interaction
- A standardised syntax and semantics for agent and agent systems-naming services
- A standardised location syntax for finding agents

4.4.1.2 The FIPA standard

The FIPA standard intends to promote interoperation of heterogeneous agents and agent services. FIPA architecture consists of the following concepts and agents:

- Agents: Each agent has an identifier that is unique in the agent environment.
- Agent platform (AP): Consists of a directory agent, a management agent and a communication channel agent.
- Directory Facilitator (DF): The directory agent is an agent that supports catalogue services to other agents. It defines an agent domain and supports the actions to register, deregister, search and modify.
- Agent Management System (AMS): This agent manages activities within an agent platform, such as the creation, deletion and migration of agents.

- Agent Communication Channel (ACC): The communication agent passes messages to agents within the platform and to other platforms.
- Agent Communication Language (ACL): The communication language used is based on the search-act theory, where messages are viewed as communicative acts intended to perform some action. ACL consists of ontologies, a knowledge representation language (KIF - Knowledge Interchange Format) and a communication language similar to KQML (Knowledge Query and Manipulation Language).

FIPA demonstrates several applications that have been implemented using their architecture ([www.: joagu@ida.liu.se](http://www.joagu@ida.liu.se)).

Although the specifications are not complete, they offer valuable guidelines to develop mobile agent systems.

4.4.2 Characteristics

In addition to the essential properties of agents as discussed in Table 4.1, mobile agents exhibit various additional unique characteristics. Lange and Oshima (1998) cite object-passing, autonomy, asynchronicity, local interaction, disconnected operation and parallel execution as unique characteristics. Braun, Eismann, Erfurth and Rossak (2001) add adaptation, communication, cooperation, persistence and goal-orientedness to this list. A mobile agent can clone itself, which implies a measure of recursion (Harrison, Chess and Kersenbaum, 1995; Horvat, Milutinovic, Kocovic, and Kovacevic, 2000). Table 4.2 summarises similar characteristics as pointed out by Nwana and Ndumu (1996b), Green et al. (1997), Horvat et al. (2000) and Schoeman (2005).

Table 4.2 Mobile agent characteristics

Property	Meaning
Mobility	It is able to transport itself from one machine to another.
Asynchronous execution	An agent has its own thread of execution, thus it does not require its sending host to suspend execution until the mobile agent returns.
Local interaction	An agent interacts locally with its host.
Disconnected operation	The agent can perform its tasks at a host, regardless of whether the network connection is open.
Parallel execution	Cloning allows more than one agent to execute a task in parallel at different hosts.
Intelligence	Agents' level of intelligence can vary, but commonly intelligence sustains autonomy by allowing the agent to decide as to where to move next.
Communication	An agent can communicate with other agents, and possibly people.
Cooperation	Agents cooperate to work within an environment, via public protocols, to reach a common goal.
Adaptation	Agents can react dynamically to changing situations by choosing an appropriate action.
Persistence	The state of the mobile agent is persistent, even if action is stopped and restarted.
Goal-orientedness	Mobile agents perform tasks on behalf of others, to achieve certain goals.
Loyalty	An agent performs computations on behalf of users.
Recursion	An agent can create child agents for sub-tasks.

4.4.3 Advantages of mobile agents

Circumstances under which the use of mobile agents is regarded as beneficial can be recognised by reviewing the advantages as well as disadvantages of mobile agents. Researchers (Green et al., 1997; Lange and Oshima, 1998, 1999,

Gawinecki, Kruszyk, Paprzycki and Ganzha, 2007) cite the following advantages of using mobile agents:

- *Agents reduce the network load* by processing data at the remote host instead of transmitting data over the network. Distributed systems often rely on communication protocols that involve interaction with the host, resulting in network traffic. Mobile agents package a conversation and move it to the host, where interaction takes place. This overcomes the limitations of a client computer with insufficient resources.
- *Agents execute asynchronously and autonomously*. Mobile devices often rely on expensive but fragile network connections. Tasks requiring a continuous open connection between two devices are commonly not economically or technically feasible. Tasks can be embedded into mobile agents, which can traverse a network and execute asynchronously and autonomously without relying on a continued connection. Thus, when network connectivity is not consistent, processing does not have to stop.
- *Agents are efficient*. Mobile agents consume fewer network resources since they move the computation to the data, rather than the data to the computation (Green et al., 1997). Vast amounts of data, such as in a weather station, can be accessed remotely, instead of moving the data over the network.
- *Agents overcome network latency* by executing and acting locally. Critical real-time systems such as robots in manufacturing processes need to respond in real-time to changes in their environment. Controlling such systems involves significant latencies. Mobile agents can act locally and execute the controller's instructions.
- *Agents encapsulate protocols*. In a distributed system, each host owns the code needed to implement the protocol to exchange incoming and outgoing data. Mobile agents can move to remote hosts to establish 'channels' based on proprietary protocol.

- *Agents adapt dynamically.* Due to the reactivity attribute defined in weak agency, mobile agents are aware of their environment and respond to changes in it.
- *Agents are naturally heterogeneous.* As network computing is often from a hardware and software perspective heterogeneous and mobile agents are usually computer and transport layer independent. Mobile agents will execute on different hardware and software systems.
- *Agents are robust and fault tolerant.* The ability of mobile agents to react dynamically to adverse situations makes it easier to build fault tolerant behaviour.
- *Agents can personalise server behaviour.* By dynamically supplying new behaviour, network entities such as routers can change behaviour when supported by intelligent agents.
- *Agents provide support for electronic commerce.* Mobile agents can build electronic markets. The mobile agent will embody the intentions, desires and resources of the participants.
- *Agents are seen as a convenient development paradigm,* since they are inherently distributed in nature and thus entail a natural view of a distributed system.

4.4.4 Disadvantages of using mobile agents

One of the main disadvantages to using mobile agents is security. This includes all security issues such as authentication of user and server, authorisation, verification agents and protecting agents against transfer (Lange and Oshima, 1999; Roth, 2004; Luck et al., 2005). Seeing that this thesis does not focus on security issues, the reader is referred to Grimley and Monroe (1999) for more information on security issues related to agent technology.

Lack of accepted standards and a pervasive infrastructure are also listed as inhibiting factors (Milojicic et al., 1999; Roth, 2004; Luck et al., 2005). Additional problems include lack of a so-called killer application, lack of proper education in

agent development, and lack of scalability or performance (Harrison et al., 1995; Samaras, 2004; Wooldridge, 2002).

Luck et al. (2005) provide a table (Table 4.3) with areas of agent technology to be addressed over different timescales. The main areas that are listed are industrial strength software, agreed standards, infrastructure for open communities, reasoning in open communities, learning technologies, and trust and repudiation. Broad and specific challenges are also identified. The following broad challenges are listed:

- Developing tools, techniques and methodologies to support agent systems developers
- Automating the specification, development and management of agent systems
- Integrating components and features
- Establishing tradeoffs between adaptability and predictability
- Establishing linkages to other branches of computing

Table 4.3 Challenges regarding agent technology (Luck et al., 2005)

Challenges	Short term	Medium term	Long term
Industrial strength software	Peer to peer Better development tools Service-oriented computing	Generic designs Libraries for agent-oriented development	Best practice in agent systems design
Agreed standards	FIPA ACL Semantic description Service-oriented computing	Libraries of interaction protocols	Tools for evolutions of communications languages and protocols
Infrastructure in open communities	Web mining Data integration Semantic web	Semantic interaction Agent-enabled Semantic web	Shared improved ontologies
Reasoning in open environments	Organisational views of agent systems	Enhanced understanding of agent societies	Automated eScience systems and other application domains
Learning technologies	Adaptation Personalisation	Evolving agents Self-organisation	Run-time reconfiguration and re-

Challenges	Short term	Medium term	Long term
	Hybrid technologies	Distributed learning	design
Trust and repudiation	Security and verification Reliability testing Self-enforcing protocols	Norms and social structures Formal methods Electronic contract	Trust techniques for coping with malicious agents

Mobile agent technologies have disadvantages and challenges. However, the benefits gained from robustness, asynchronous and autonomous functioning, speed and functionality far outweigh the disadvantages.

The main reasons for using mobile agents can be summarised as follows:

- Intermittent connectivity, slow networks, lightweight devices and mobile computing, as well as offline processing with unreliable and/or limited capacity.
- Asynchronous execution, autonomy and persistence or distributed retrieval or dissemination of information (Sunsted, 1998; Lange and Oshima, 1999).

Domains where mobile agent technology can make a positive contribution include but may not be restricted to are as follows (Lange and Oshima, 1999; Kendall, Krishna, Suresh and Pathak, 2000, Gawinecki et al., 2007):

- *Telecommunication and networking*: Support and management for advanced telecommunication services use dynamic network reconfiguration and user customisation. Mobile agents can act as the glue to keep these systems flexible but effective (Lange and Oshima, 1999; Kendall et al., 2000).
- *Electronic commerce*. A commercial transaction may require real-time access to remote resources. Different agents with different goals and strategies may each support these. Agents embodying the intentions of their creators, and acting and negotiating on their behalf constitute a suitable application for agent technology.

- *Mobile computing:* Laptops and other mobile wireless technologies are still limited in terms of bandwidth usage. Mobile agents allow for increased performance and can proceed executing when connections fail.
- *Distributed information retrieval and dissemination systems:* Agents can retrieve large amounts of information from various distributed locations. Agents can also disseminate information such as news and software updates to specific persons.
- *Distributed computing:* Processing of data can be done closer to the data source instead of moving large amounts of data to the processor. Only the result, rather than large amounts of data, can then be returned.
- *Distributed management:* Mobile agents can be designated management tasks closer to where they are needed in a distributed area, thus reducing bandwidth usage.
- *Workflow applications and groupware:* Mobile agents are suited to support the flow of documents between co-workers. Agents provide a degree of autonomy as well as mobility to these workflow documents.
- *Monitoring and notification:* An agent can monitor a certain information source without being dependent on the system from which it originates. The asynchronous nature of the agent supports this function, as the agent can wait for certain kinds of information to become available.
- *Personal assistants:* Agents can perform on behalf of their creators over a distributed network. Remote assistants work independent of their network connectivity. For example – when a meeting is to be scheduled, a user can send an agent to interact or negotiate with team members or agents of team members to arrange the meeting.

4.5 CONCLUSION

This chapter was devoted to a discussion of agent technology. The background to agent technology was sketched and emerging trends were reviewed as drivers for agent technology. Software agent technology was investigated, including various classes of software agents and specifically mobile software agents. The

advantages and disadvantages of using agent technology were explored and standards for mobile agents were considered.

Despite the lack of complete standards documentation, mobile agent research continues. This is clear from the attention that international mobile agent conferences are receiving, as well as from the recent launch of the *Journal of Mobile Information Systems*, which includes mobile agents as topic (Tanair, 2005).

Mobile agent technology can therefore be seen as a *new paradigm that may be used to support the SPM process*. The following table lists the unique requirements of SPM and how agent technology may address these.

Table 4.4 SPM features to be addressed by agent technology

SPM	Software agent technology
Changing environment of SPM systems leads towards a <i>complex distributed environment</i> .	Agents allow distribution and communication over a geographical area irrespective of the geographical location. Agents overcome network latency by executing locally, thus reducing network load (Lange and Oshima, 1998, 1999). Parallel execution enables tasks to be executed in parallel at different workstations.
SPM distributed environments may require and incorporate mobile devices and <i>fragile network connections</i> .	Agent systems can incorporate large network systems and mobile devices. Tasks can be embedded into mobile agents, which can traverse the network and execute asynchronously and autonomously without relying on a continued connection (Harrison et al., 1995; Lange and Oshima, 1998).

SPM	Software agent technology
A distributed environment requires a high level of <i>collaboration and cooperative problem solving</i> between teams and team members.	Teams of agents can coordinate actions towards a similar goal in a distributed environment. Software agent technology provides a natural metaphor for support in a team environment, where software agents can traverse the network in order to monitor and coordinate events (Wooldridge, 2002). Communication and cooperation is strongly supported by agent teams.
Distributed environment results in virtual, <i>dynamically changing</i> collaborative teams.	Agents adapt dynamically to changes. They are aware of their environment and can respond to changes in it. Agents have the computational mechanisms for flexibly forming, maintaining and disbanding organisational structures (Jennings, 2001).
Collaboration between team leader and distributed team members requires continuous <i>control, monitor and measurement</i> . (Invisibility aspect)	Agents are excellently suited to control, monitor and measure elements in a distributed environment (Braun et al., 2001).
A distributed environment requires <i>heterogeneous technology</i> and databases that have to interact and share information.	Agents are naturally heterogeneous, and mobile agents can execute on different hardware and software platforms (Impey and Forester, 2003; Lange and Oshima, 1999).
The changing SPM environment requires <i>flexibility and conformity</i> of the system.	Agents adapt dynamically to changes in their environment; thus this feature will be excellently supported (Nwana and Ndumu, 1996a; Kotz, Jiang, Gray,

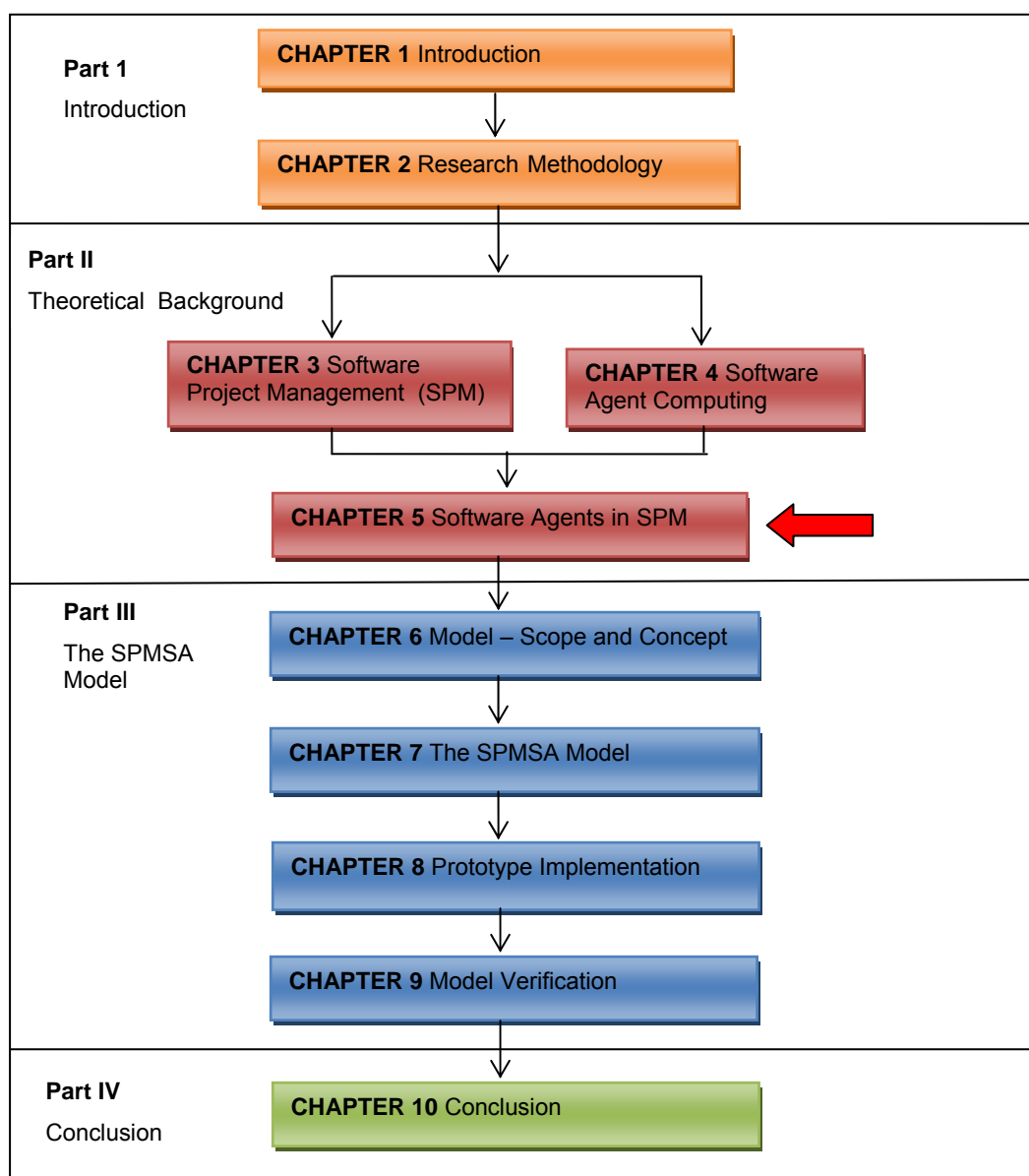
SPM	Software agent technology
	Cybenko and Peterson, 2000).
Virtual software project teams over dispersed environments need to <i>access information and documents</i> .	Agents support the distributed retrieval and dissemination of information and documents and can automate routine tasks (Maes, 1996; Green et al., 1997).

Table 4.4 illustrates that software agent technology is indeed suitable for addressing the various unique features of SPM. It can therefore be concluded that software agent technology provides a suitable framework for supporting and possibly enhancing SPM processes in a complex distributed environment. The need for the flexible management of ever-changing organisational structures such as those dealt with in SPM is suitably addressed by the computational mechanism of agent systems (Jennings, 2001). Agent behaviour as stated in Table 4.4 can be used to support the individual team members in numerous tasks, such as coordination and cooperation with team members, document retrieval and distribution, workflow monitor and control, scheduling and organisation of meetings, reminders for tasks and overdue dates or deliverables.

Various development approaches as well as environments for agent systems exist. In the next chapter the development environments of agent systems will be investigated. The researcher will also look at software agents used in different applications and explore the possibilities of using this paradigm to support SPM processes.

CHAPTER 5

5 SOFTWARE AGENTS IN SPM



5.1 INTRODUCTION

In the previous chapter the basic concept of agents – and more specifically *mobile* software agents – was explored. It was concluded that agent technology can be utilised to support and possibly enhance the entire SPM environment, so as to address shortcomings and project failures in this area.

Mobile software agent *development* and *implementation* will be discussed in this chapter. Agent application development environments have been generated to support the user in his/her process of designing and implementing an agent-based system. The purpose of this chapter is to regard the process of agent development, as well as the factors that support this environment. Finally, Chapter 5 also investigates agent-driven applications that have been developed with specific focus on the agents used in SPM.

Wherever reference is made to a mobile agent throughout the remainder of this thesis, it should be read as referring specifically to a *software* mobile agent.

5.2 MOBILE AGENT DEVELOPMENT

Developing *agent* applications is a relatively new software engineering paradigm and although a variety of approaches for development exists, a standard has not been set and proven (Iglesias, Garijo, Gonzalez and Velasco, 1998; Zambonelli, Jennings, Omicini and Wooldridge, 2001). Existing methodologies are/were taken and either adapted or extended to support agent-based development. Examples of these are existing object-oriented (OO) methodologies, knowledge engineering, formal specification languages, as well as new methods such as using patterns and components. The adoption of agent technologies has not yet entered the mainstream of commercial organisations, and the range of applications is limited to a small number of industrial sectors (Luck et al., 2005). These include automated trading in online markets such as for financial products and commodities; simulation and training applications in defence domains; network management; control system management in industrial plants such as steel works; user interface and local

interaction management in telecommunication networks; schedule planning and optimisation in logistics, and supply-chain management (Luck et al., 2005). Which options, i.e. methodologies, and application development environments are then available to design and implement mobile agent systems? The following sections attempt to answer this question.

5.2.1 Mobile agent development methodologies

Software development methodologies usually consist of a modelling technique and a development process. Models are intended to formalise the concept of the system as they are defined on an abstract level and then extended to become more detailed, concrete and specific. An analysis and design methodology assists in gaining an understanding of and then designing the system. This thesis focuses on compiling *a model of SPM processes supported by an agent framework to enhance the SPM processes*. Thus, the methodology and process to implement an agent system will only be regarded briefly and in the context of an agent system.

Wooldridge (2002) specifies two broad groups of approaches to analyse and design agent-based systems:

- Agent systems that adapt to object-oriented development and that either extend existing OO methodologies or adapt OO methodologies to the purpose of agent-based systems (Burmeister, 1996; Kendall et al., 2000; Wooldridge, et al., 2002).
- Agent systems that adapt to knowledge engineering or other techniques such as formal specification techniques (Kefalas, Holcombe, Eleftherakis and Gheorghe, 2003; Luck et al., 2005).

There are various examples of the above approaches, namely the AAll methodology defined by Kinny and Georgeff, (1997); Odell's (2001) adaptation of UML to agent development; the Gaia methodology of Wooldridge et al. (2000); the use of specification language Z to specify agent systems (Luck et al., 2005); as well as the Cassiopeia methodology designed by Collinot, Drogul and Benhamou (1996).

Debenham and Henderson-Sellers (2003) propose extensions to the OPEN (Object-oriented Process, Environment and Notation) framework to design agent-based systems.

An additional methodology has emerged that is specifically tailored to address the development of agent-based systems, namely *agent-oriented analysis and design*. Agent-based systems have many unique factors, thus it is inevitable that it should be addressed by a unique analysis and design methodology, specifically designed for this paradigm. An example is the Hermes system that proposes an approach for designing agent interactions in terms of interaction goals (Cheong and Winikoff, 2005). Design goals are then mapped into collections of plans.

Ongoing research is conducted into the process of systems analysis and design of agent-based systems. Currently there is no single unified and unique agent-orientated methodology, i.e. a standard has not been adopted yet (Schoeman, 2005). Methodologies that have been developed vary, for example Gaia (Wooldridge, et al., 2000) and SODA (Omicini, 2001), which support only analysis and design phases, and PASSI (process for agent societies specification and implementation) which supports the entire spectrum of the life cycle (Cossentino, Burrafato, Lombardo and Sabatucci, 2002).

However, whichever methodology is followed, the process of developing and implementing software agents will also involve a phase for the *analysis and design* of the agent system, and thus a brief look at these phases is necessary to put them in the context of this paradigm (Schoeman, 2005).

During the software development process at least four basic phases are usually executed in either sequential or concurrent order. These are the requirements analysis phase, design phase, implementation phase and testing phase. Each phase results in various diagrams, depending on the paradigm or methodology used. For example, the models supporting the requirements analysis phase use

case diagrams, domain models and system sequence diagrams, while the models supporting the design phase use the design class diagram, interaction diagrams and package diagrams (Satzinger, Jackson and Burd, 2004). The phases of development may continue sequentially (as in the waterfall method), or iterations will repeatedly take place (V-model, prototyping, and iteration method).

For the purpose of this study, the requirements analysis phase and the design phase will be considered in more detail.

5.2.1.1 Requirements analysis phase

During the requirements analysis phase the problem domain is investigated and the problem(s) identified. The analyst concentrates on what the problem is and identifies functional and non-functional requirements. Functional requirements identify *what* the system must do (use cases are used) and non-functional requirements may constitute themselves through the constraints on the system, i.e. security. Thus, the system's goals and requirements must first be defined, as is the case in traditional systems analysis methodologies.

Regardless of the model, technique or methodology that is used to design and implement an agent system, the analysis process should result in a conceptual model to describe both the conceptual agent model and the conceptual group of agents to be implemented. This model states *what* the system has to do, not *how* it is to be done. Thus, to summarise, the following must be identified (Zambonelli et al., 2001):

- A description of the requirements of the system
- A model to describe external interaction with the system and individual agent responsibilities/tasks
- An agent social model for global interaction

5.2.1.2 Design phase

During the design phase the developer creates the solution to the problem based on the outcome of the requirements analysis phase (Schoeman, 2005). The main focus of this phase is on *how* to resolve the problem. This is traditionally identified by an architectural as well as a detail design. The three main methods for developing agent systems are based on extensions and adaptations of either OO methodologies or knowledge-based methodologies, or on agent-based approaches (Zambonelli et al., 2001; Schoeman, 2005).

The following step will be to *implement* the agent system by using either a programming language or a development environment, as will be discussed in the following section.

5.3 MOBILE AGENT IMPLEMENTATION

The design models developed during the agent analysis and design phase need to be implemented by translating them to program code. This can be done either by using a programming language and developing the entire agent system from scratch, or by using an agent development environment that provides the programming constructs to implement agent concepts. To build sophisticated software agents from scratch can be very difficult and time consuming. Hayes-Roth and Amor (2003) and Luck et al. (2005) recommend that software developers without extensive experience in agent development use an agent construction toolkit to build software agents.

The form and components of toolkits vary from integrated development environments to basic middleware providing some networking capabilities. According to Luck et al. (2005) agent toolkits will typically offer the following:

- Facilities to enable the development of individual agents, and their interface to the environment.
- Coordination and communication mechanisms with regard to high-level and low-level services.

- Management services to monitor and debug agent applications.
- Software to assist with the development process.

5.3.1 Agent development environments

Mobile agents interact, move and perform various autonomous tasks. Mobile agents need an environment in which they can execute, migrate and communicate. This environment is addressed in different terms by various authors, i.e. it is referred to as a “platform” (Tahara, Ohsuga and Honiden, 1999; FIPA, 2003), “framework” (Feridun and Krause, 2001), “architecture” (Gschwind, Feridun and Pleisch, 1999; Tahara et al., 1999; Wong, Helmer, Naganathan, Polavarapu, Honovar and Miller, 2001) or “infrastructure” (Aridor and Oshima, 1998).

An agent *platform* can be seen as the computer hardware as well as software available for agent development (Caudron, Groote, Van Hee, Hemerik, Somers and Verhoeff, 2004). *Agent frameworks* are programming toolkits or development toolkits for constructing agents such as D’Agents (D’Agents: Mobile agents at Dartmouth College, 2002), Grasshopper or JADE. *Agent architecture* views agents as reactive/proactive entities and conceptualise agents as perceptive, reasoning and active components (Schoeman, 2005). *Agent infrastructure* provides the rules that agents follow, and deals with ontologies, communication protocols, communication infrastructure and interaction protocol.

5.3.1.1 Agent platforms

In this thesis an agent environment that allows for the proper functioning of agents is referred to as an *agent platform*. Key aspects of an agent platform include support for communication, interoperability, security and mobility. An agent platform must have an agent management system to support communication, interoperability, security and mobility (Van Zyl, 2005). Agent platforms may also provide a set of standards and development tools to aid in the design, construction, management and maintenance of agents.

Most existing mobile agent platforms are based on either scripting languages or Java. The latter is currently the language that is most commonly used to program mobile agents (Green et al., 1997; Wooldridge, 2002; Luck et al., 2005).

Various agent development platforms have been developed that address the technical issues of mobility of an agent in different ways. For example, in Telescript mobility is employed by transmitting both the agent and its state. The state includes a program counter that 'remembers' the value and resumes when it reaches the new destination (Wooldridge, 2002).

In order to *implement* an agent platform, the following minimum criteria must be met (Luck et al., 2005; Van Zyl, 2005; Schoeman, 2005):

- Development support for mobile agents must be provided. The agent platform must provide support for agents to communicate with each other in order to extract information, give information or check information. Allowing agents to move to the location where interaction will take place means that network delays, disconnection or downtime will not affect the communication or negotiation process.
- The platform must be operating system independent. In order to allow agents to move from one system to another, regardless of the underlying operating system, will provide greater usability to the agent community. Broader usage is essential for large applications and distributed environments.
- The agent platform must support standards such as FIPA or the mobile agent system interoperability facility that was discussed in the previous chapter (Van Zyl, 2005). Standards allow for agent communication and mobility; thus, following a set standard allows for further expansion and interaction between systems.
- The development platform must support Java. Java has become a *de facto* standard for the programming of mobile agents (Luck, 2005; Chmiel, Tomiak, Gawinecki, Kaczmarek, Szymczak and Paprzycki, 2004b). Java not only

supports weak mobile agents through serialisation, but also supports operating system independence.

Agent platforms that support the above criteria include Aglets software development kit (ASDK); Bee-gent; Comtec agent platform; Grasshopper; Java agent development environment (JADE), and Tryllians's agent development kit (ADK).

The Java agent development environment (JADE) platform is shown in Figure 5.1. In this platform, agent development is supported by an agent management system, directory facilitator and message transport system. The agent management system enables agent mobility, interoperability, communication, as well as the security of agents. The directory facilitator provides a yellow-page service to monitor the agents, while the message transport system enables agent message transportation by communicating with other agent platforms.

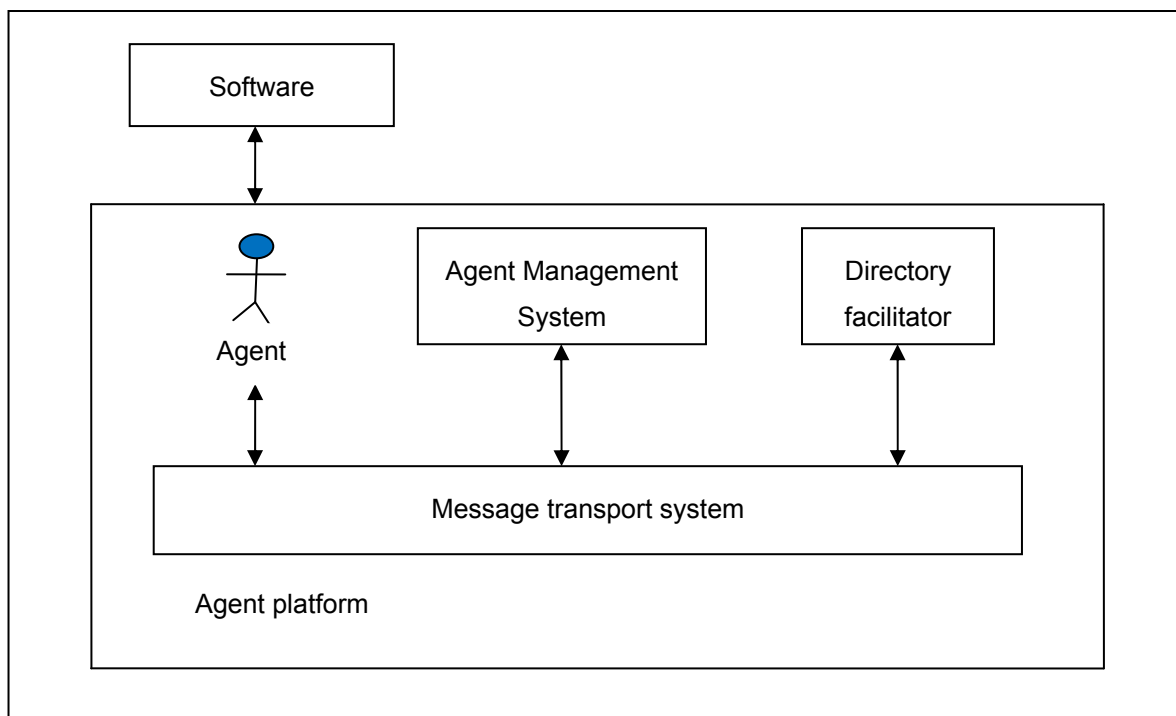


Figure 5.1 JADE Agent platform

Other prominent development environments not necessarily based on Java are D'Agents (formerly known as AgentTCL), Telescript and Tacoma, all based on TCL. Table 5.1 lists some of the available agent platforms (adapted from Altmann, Gruber, Klug, Stockner and Weippl, 2000 and Giang and Tung, 2002). It also supplies the name and author of each platform, as well as information on its support for standards, security and communication.

Table 5.1 Platforms for agent development

	Author	Features
Agenttalk	NTT Ishida	A coordinated protocol description language for multi-agent systems. AgenTalk allows protocols to be defined incrementally and to be customised to application domains by including an inheritance mechanism.
Agent Development Kit (ADK)	Tryllian BV	This commercial kit supports P2P XML-based communication. Supports weak mobility. A commercial kit not suitable for academic use.
AgentTool	Kansas State University	The AgentTool allows agent designers to formally specify the required structure and behaviour of multi-agent systems. Supports authentication and access restriction. Full support for communication. Supports weak mobility.
Aglets Software Development Kit (ASDK)	IBM	An aglet is a Java object that can move from one host on the Internet to another. The aglet can halt execution, dispatch to the remote host and resume execution. The aglet's state and code is transmitted. Open source, free.
AgentBuilder		An integrated tool suite for constructing intelligent software agents. Tools for managing the development process, analysing the domain of agent operations, designing and developing networks of communicating agents. The runtime system includes an agent engine that provides an environment for execution. Full support for communication: Knowledge and Manipulation Language (KQML)

	Author	Features
Grasshopper	IKV++ Technologies AG++	Good security support for certificates, encryption and authentication. Full support for communication, MOTIF and FIPA standards. Strong mobility support, NOT open source project, thus access to the working of the system is limited.
Java Agent Development Environment (JADE)		JADE constitutes a software framework to develop agent-based applications. A comprehensive set of system services and agents is provided. JADE can be considered agent middleware. Weak security, full support for communication. Complies to FIPA standards, open source, free.
Zeus	British telecommunication labs	Zeus is a collaborative agent-building environment and component library written in Java. Each Zeus agent consists of a definition layer, organisational layer and a co-ordination layer.

For a summary of technology refer to Green (1997), Altmann et al. (2000) and Giang and Tung (2002).

5.4 APPLICATIONS OF MULTI-AGENT SYSTEMS

Various applications of agents have been implemented. As was stated earlier, Luck et al. (2005) suggest that the adoption of agent technologies has not yet entered the mainstream of commercial organisations and the range of applications is limited to a number of industrial sectors. However, Ganzha and Paprzycki (2008) are actively researching agent technology and are busy implementing systems such as a travel support system (Kruszyk, Ganzha, Gawinecki and Paprzycki, 2007) and an e-commerce system (Vukmirovic, Gawinecki, Kobzdej, Ganzha and Paprzycki, 2007).

To develop a mobile agent system, an adaptable and flexible framework is needed. It should include multi-agent features that enable the developer to set up the distributed application, as well as an appropriate level of reasoning.

Applications of agent systems are divided by Wooldridge (2002) into two main categories, namely distributed system agents (with the emphasis on multi-agent systems) and personal assistant agents (with the emphasis on individual agents).

Examples of agent applications include agents for e-commerce, agents for information retrieval and management, agents for networking and the Internet, agents for workflow and business process management, as well as agents in project management (Wooldridge, 2002; Luck et al., 2005).

5.4.1 Agents for Electronic Commerce

E-commerce systems (of which amazon.com is one of the best-known examples) will typically allow the user to browse an online catalogue of products, select some and then purchase these products using a credit card (Wooldridge, 2002). Agents have improved on these systems by automating some of the buyer's behaviour. They have been used as comparison shopping agents, where the agent obtains information related to available products, examines various merchants, negotiates for the buyer and purchases the product (Wurman, 2001). The InAMoS project proposes a mobile agent to represent the user in the market place (Luck et al., 2005). The Jango system (Wooldridge, 2002) also represents a comparison shopping agent system.

5.4.2 Agents for Information Retrieval and Management

Distributed, semi-structured information resources such as the World Wide Web provide enormous potential for the accessing of information. Agents can be used for searching the internet and filtering information to prevent 'information overload'. For example, Pattie Maes from the MIT lab developed a number of prototypical systems that could carry out these types of tasks. MAXIMS, an electronic mail filtering system can prioritise, delete, forward, sort and archive mail messages on behalf of the user (Maes, 1994). Another example is the NewT system, a Usenet news filter (Maes, 1994) where the agent filters news as an extension of the user's interest, and then searches and proposes such news items for the reader to read.

MagicCap is an intelligent personal communication system that uses mobile agent technology in Telescript to allow different forms of communication to intelligently interact with the user, irrespective of his geographical location (Green et al., 1997). A number of studies have been conducted on information agents, including aspects such as extracting or including information from different sources (Gruber, 1993). The SoFAR system represents an agent framework for distributed information management (Moreau, Zaini, Cruickshank and De Roure, 2003). Carnot (Huhns, 2002) allows pre-existing and heterogeneous database systems to work together to answer queries that fall outside the scope of the individual database. Obudiye, Kocur and Weinstein (1997) developed an agent-based information retrieval system called SAIRE.

Related work is being done by the computer-supported cooperative work (CSCW) community. Interested readers may consult the work of Gaeta and Ritrovato (2002) and Greif (1994) for more detail.

5.4.3 Agents for Network and Internet

The use of mobile agents in network management and in the telecommunications area has been recognised and promoted. The IBM Agent Meeting Point for Mobile Communication concerns itself with the creation of a framework to implement secure, remote applications in large public networks with several mobile devices such as laptops and PDAs (Green et al., 1997). An agent meeting point is central to this concept. The Magna mobile agent system that was developed by GMD Fokus and the Technical University of Berlin targets the service scalability problem in intelligent networks by trying to incorporate Remote Procedure Calls as well as mobile agents for providing customised telecommunication services (Green et al., 1997). Another example of mobile agents used for advanced network management is the Perpetuum Mobile Procura Project. In this application, mobile agents are used to overcome the problem of legacy issues, explore new intelligent distributed management techniques and deliver a Java-based platform for network management. A collection of articles on agent systems (Programming Multi-agent

Systems), provide more information for the interested reader (Eds.) Bordini, Dastani, Dix and Seghrouchni (2005).

5.4.4 Agents for workflow and business process management

Workflow and business process control systems are of increasing importance in computer science. Workflow systems aim to automate the processes of a business, making sure that the correct tasks are sent to the correct people at appropriate intervals, and typically ensuring that a specific document flow is maintained and managed in an organisation (Müller, Bauer and Friese, 2004). In the agent system ADEPT a business organisation is modelled as a society of negotiating service-providing agents (Jennings, Sycara and Wooldridge, 1998).

5.4.5 Agents used in project management

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 (Balasubramanian, Brennan and Norrie, 2001; Chen et al., 2003). In this section, we consider how agent technology is currently deployed, specifically 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 and Welch, 2004). 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 and Davis, 2003). However, work on the use of agents in project management has been performed to address certain aspects pertaining to SPM, *but not to address the total environment* (Maurer, 1996; O'Connor and Jenkins, 1999; Sauer and Applerath, 2003).

5.4.5.1 Scheduling

The first application is intended for the broader project management environment, and is not specific to the SPM environment. Nevertheless, this example is mentioned as it applies agent technology to *scheduling tasks*, which are common to both environments, and it is focused on in the distributed environment. In a recent work, Sauer and Applerath (2003) presented an approach that involves using a generic agent framework to support the scheduling tasks within the supply chain in the PM environment. The framework allows for the consistent design of agents that reside on several levels of the organisation. To prevent communication overhead (found in earlier multi-agent systems), agent *teams* are formed. All the agents in a team then collaborate to solve a specific scheduling task on a particular level. Furthermore, every agent (in its personal capacity) is also responsible for a specific schedule (the schedule of the resources that it represents). Therefore each agent is provided with the scheduling knowledge that is necessary to create or maintain the schedule without contacting the other members of the team. The focus of this application is primarily on time management and certain aspects of the communication management function.

Agents are used with great success as planners and schedulers in supply chain management and industrial systems. Examples of these are the Daewoo system produced by Metra Corp where task, resource and service agents schedule the press shop at Daewoo Motors' integrated automobile production facility in Korea (Wu and Simmons, 2000). The Daewoo shop supplies body parts for five different car models. An agent is assigned to each traditional manufacturing function, such as order acquisition, logistics, scheduling, resource management. Agents are also commonly assigned to physical entities in the system. The AARIA (Autonomous Agents for Rock Island Arsenal) ontology is used and manufacturing processes occur when the flow of the parts and resources intersect at a unit process (Parunak, Baker and Clark, 1997).

5.4.5.2 Planning and resource management

Maurer (1996) proposed a system (the *CoMo Kit*) in which methods and tools were developed to plan and manage complex workflows, especially in design domains. According to this system, tasks can be decomposed into subtasks and for every task several alternative decompositions (methods) can be defined. Every task is associated with a set of agents, humans or computers that are able to solve it. The problem-solving process, for example the application of methods to tasks, is distributed via a local area network. The proposed system uses agent technology as a tool for planning, coordinating and designing process execution. This approach follows a centralised black-box agent approach. The system architecture consists of a *modeller* that does project planning; a *scheduler* that supports project execution and manages information produced; and an *information assistant* that allows access to the current state of the project. During SPM, the *modeller* gathers information through interaction with the project manager or other stakeholders, and as a result presents a model of this information to the *scheduler* as input. The scheduler then manages agendas that contain the tasks to be carried out by an agent. To work on the task, the agent can access all relevant information (using the information assistant) for solving the problem. Maurer's solution (1996) is applicable to scope management, time management and, to a certain extent, the communication management function. Research is also conducted on resource management in virtual organisations (Szymczak, Frackowiak, Ganzha, Gawinecki, Paprzycki and Park, 2007) and plan tracking (Wu and Simmons, 2000).

In another example targeting this environment, O'Connor and Jenkins (1999) propose an intelligent assistant system to support the project team during planning, scheduling and risk management.

Joslin and Poole (2005) adapt a simulation-based planning algorithm to the problem of planning for SPM. Simulation techniques offer support for modelling the way in which agents may behave in project management and the manager might adapt the project plan based on the project status at future points. Resource allocation and

task selection are targeted by this simulation. The algorithm for resource allocation is run at the beginning of each simulation period.

5.4.5.3 Control and monitor

Software agents are used to control and monitor activity execution at various sites in an open source platform that supports distributed software engineering processes in a development as part of the GENESIS project (Gaeta and Ritrovato, 2002). Software agents are used 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.

5.4.5.4 Risk analysis

Korb, Engel, Boesecke and Eggers (2003) apply a basic level of risk management in clinical research to implement the robot system RobaCKa for craniotomies. A systematic approach was implemented to support fault-free design, error detection and quality assurance in the design of the robot system. The system was implemented and tested, while further clinical investigations will be carried out in the next two years.

Rigaud and Guarnieri (2002) developed the AUDI@R system that aims to prevent technical risks in small and medium enterprises. The virtual organisation targeted in this application is viewed in terms of three sub-systems: the first based on activities and internal actors of the company; a second that integrates companies evolving in the same field, and a third that includes all the companies in the same geographic basin. The first sub-system is the one that concerns safety and reliability issues. Seven user profiles are identified according to company functionality, namely organisation, documentation, environment, human, production means and manufactured product. When the contractor wishes to diagnose his company, he informs the co-worker through the communication interface. A questionnaire is then

posted to users concerning security, maintenance and suitability between production means and manufactured products, after which the results are returned to the contractor. This model is based on grid computing and focuses on the system's heterogeneity and communication architecture. However, it only targets one aspect of the development cycle, namely risk (Rigaud and Guarnieri, 2002). Although Boehm laid some foundations for risk management and various related discussions (e.g. Charette, 2002) are found in the literature, formal risk analysis is rarely an integrated part of project management.

5.4.5.5 Quality assurance

Leung and Poon (1999) developed a multi-agent environment framework (AUTOQ) that aims to support software quality assurance. Emphasis is placed on process and product assurance. An interface agent assumes the role of project manager to direct requests from the user to agent testing, audit, review, defect and support components. The user provides input data to the interface agent, and it is sent to the information agent where it is tested and archived in a database. Task agents analyse the reported results, send results to the project managing agent and archive the results.

The other components, namely audit, review, defect and support components execute on the same basis by accepting input from the user, analysing and then reporting and archiving results. This system has been developed using Java and C++. It also targets only one of the eight functional areas of SPM.

This discussion serves to show the application domains where agents may be utilised. This is by no means a complete list of all agent applications, but is intended to show the possibilities of this paradigm.

5.5 CONCLUSION

This chapter was devoted to a discussion of mobile agent development and implementation. Various applications using agent technology were discussed, with specific emphasis on the SPM area to show the possibilities of using this paradigm.

Agent technology has been more commonly applied to areas such as network and system management (Kendall et al., 2000), decision and logic support (Burstein, McDermott, Smith and Westfold, 2000), interest matching (Object Management Group, 2000), data collection in distributed and heterogeneous environments, searching and filtering, negotiating and monitoring (Venners, 1997; Kruszyk et al., 2007; Ganzha, 2006). Agents are not commonly used in SPM applications and are typically constrained to one or two of the core and facilitating functions such as planning, scheduling or communication. In the previous chapter it was concluded that software agent technology can address the unique features of SPM, and that this technology is well suited to the dynamically changing environment of SPM. The software agent paradigm, including its concepts and techniques, are well suited to develop complex, dynamically changing, distributed systems (Jennings, 2001). From this chapter it becomes evident that although agent technology has indeed been applied to the SPM environment, it has not been applied to the whole spectrum, i.e. *to all core and facilitating functions of SPM*. This is a limitation of current software agent applications. Supporting and enhancing the whole spectrum of SPM processes by a software multi-agent system could provide software project managers with significant advantages over using contemporary methods (Jennings, 2001). The potential advantages that will result from this approach become clear through the increasing number of deployed agent applications in other application areas (Gawinecki et al., 2007; Ganzha et al., 2006; Sauer and Applerath, 2003; Jennings, 2001).

This chapter concludes Part II (comprising chapters 3, 4 and 5), which provides the theoretical knowledge about the areas under discussion, namely SPM and mobile agent computing. Part III will be devoted to presenting the proposed model that will

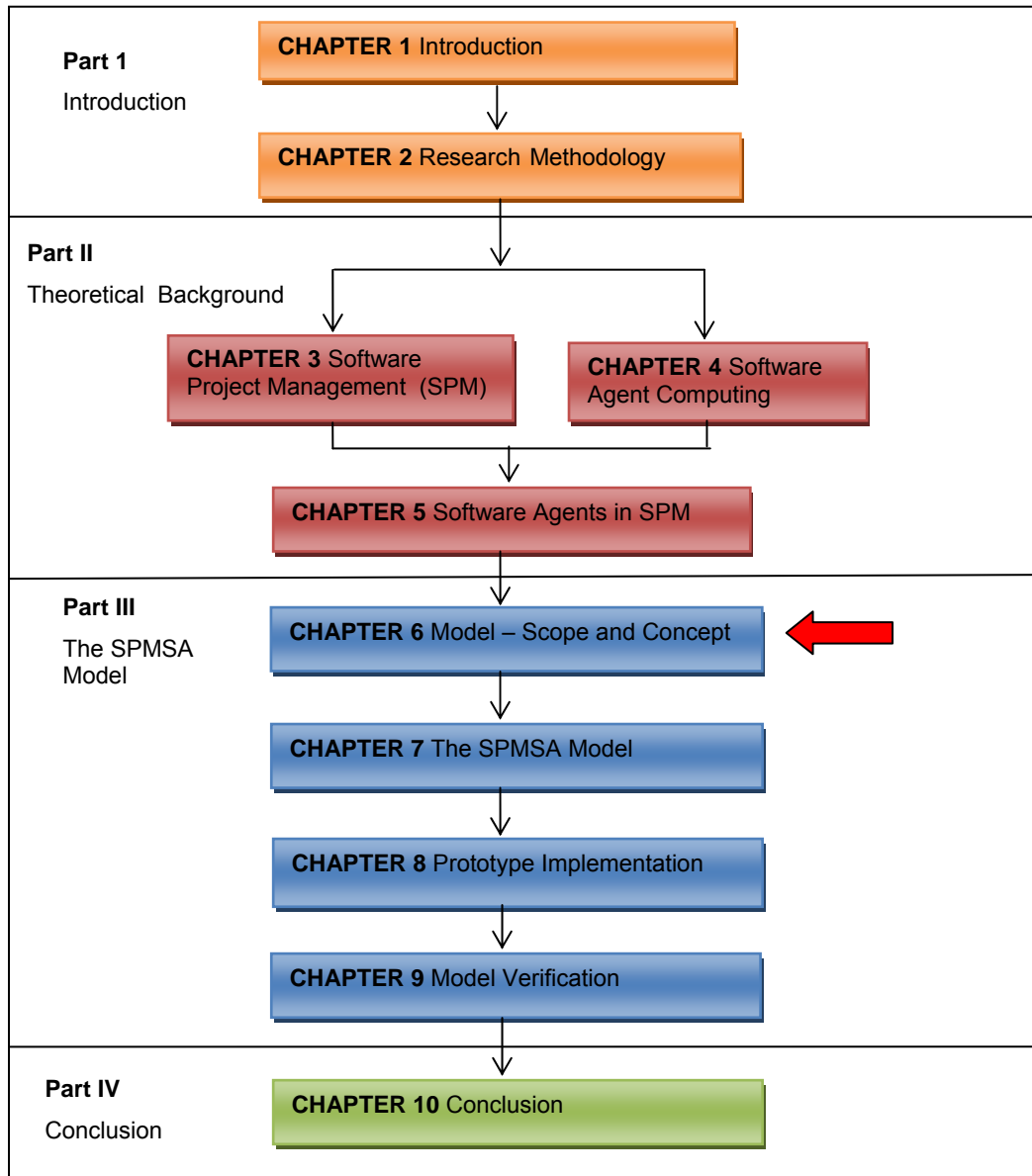
support SPM processes. A comprehensive agent framework that forms part of the proposed model will be compiled to support these processes. The next chapter provides an overview of the scope and concept of the said model.

PART III

THE SPMSA MODEL

CHAPTER 6

6 MODEL – SCOPE AND CONCEPT



6.1 INTRODUCTION

Part II established the theoretical framework that contains theoretical knowledge on SPM and software agent computing. In this part it was initially established that new approaches to address shortcomings in software development projects are needed. Software agent technology was therefore investigated to determine if agent technology would be suitable to address SPM problems in a distributed environment. It was concluded that software agent technology is particularly suitable for addressing the unique requirements of SPM.

Part III, in turn, is devoted to introducing and developing a model for SPM where the SPM processes are supported by software agents. The model entitled “SPMSA” (Software Project Management supported by Software Agents) aims to enhance the SPM processes by addressing the intrinsic unique aspects of SPM. The comprehensive framework of agents that forms part of the SPMSA model will be construed to support the entire SPM process and thereby aim to eliminate failure and address shortcomings in this environment. This model will be unique in that it aims to support and enhance the *entire* environment of the SPM arena, and not only a section of it. Current software agent applications target only a section of this environment, for example, planning or resource management. Software agent technology, as opposed to other programming paradigms, not only provides support to the dynamically changing environment of SPM, but also to its complex heterogeneously distributed environment. Furthermore, regular tasks may be automated and intelligence added to further support and enhance the workload of each team member.

The first part of this chapter will present the entire scope of the proposed model and includes both SPM and agent computing. The aim is to place the problem in context for the reader and illustrate areas of agent support to SPM processes. The latter part of this chapter will provide a conceptual view of the proposed SPMSA model. The detailed SPMSA model will be compiled and discussed in Chapter 7.

6.2 SCOPE OF THE MODEL

The proposed SPMSA model supports *all* of the SPM processes involved, as opposed to current agent applications in SPM that support only *part* of the SPM environment. The scope of the model is defined in terms of the SPM processes it supports, as well as the type of agents that support these processes. Figure 6.1 depicts the scope of the model.

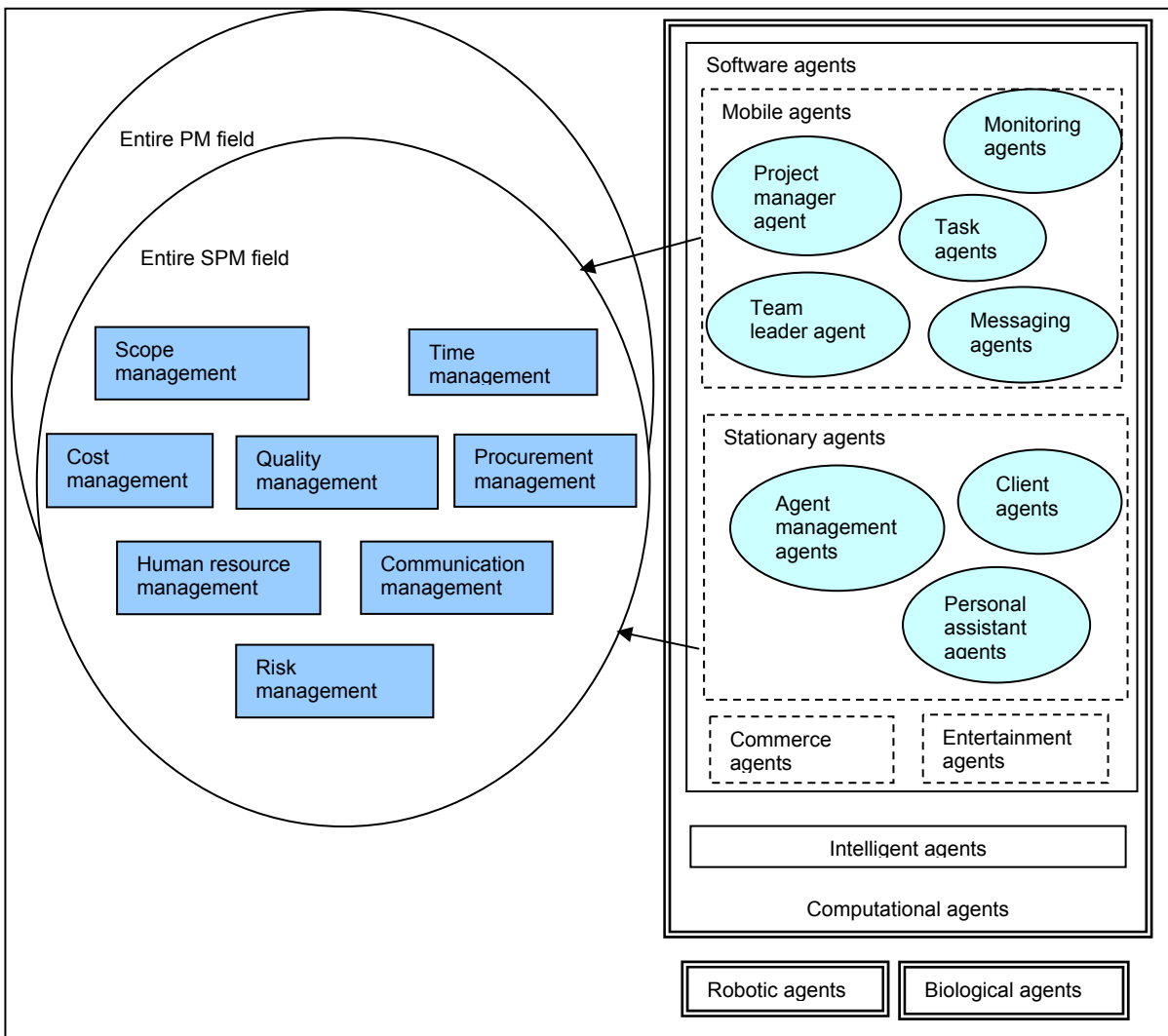


Figure 6.1 Scope of the proposed model

The left-hand side of Figure 6.1 depicts the PM field and indicates that SPM has developed into a research field of its own. The proposed model includes *all* the SPM processes indicated by the shaded blocks, namely scope management, time management, cost management, quality management, procurement management, human resource management, communication management and risk management.

The right-hand side of Figure 6.1 depicts agent computing, which comprises robotic agents, biological agents and computational agents. Computational agents can be subdivided into intelligent agents and software agents, while software agents consist of mobile agents, stationary agents, commerce agents and entertainment agents. The mobile agents that form part of the SPMSA model are shaded, namely project manager agent, monitoring agents, the team leader agent, task agent and messaging agents. The stationary agents that form part of the proposed model are also shaded. These are client agents, agent management agents and personal assistant agents. The commerce and entertainment agents fall outside the scope of the proposed model.

6.3 CONCEPT OF THE MODEL

The main goal of the proposed SPMSA model will be to support the teams and individual team members in the SPM environment while executing their tasks, and in this way to enhance the complete SPM environment. The team leader, teams and individual team members will be supported during each process of software project management to simplify the environment, eliminate the complexities, enhance coordination and communication, implement dynamic changes in the system, support task scheduling, and enhance all processes. Figure 6.2 explains the concept of this process of support.

Figure 6.2 also comprises two basic concepts, namely the *phases of software development for each SPM key function*, and the *software agent framework* that will support each key area of SPM.

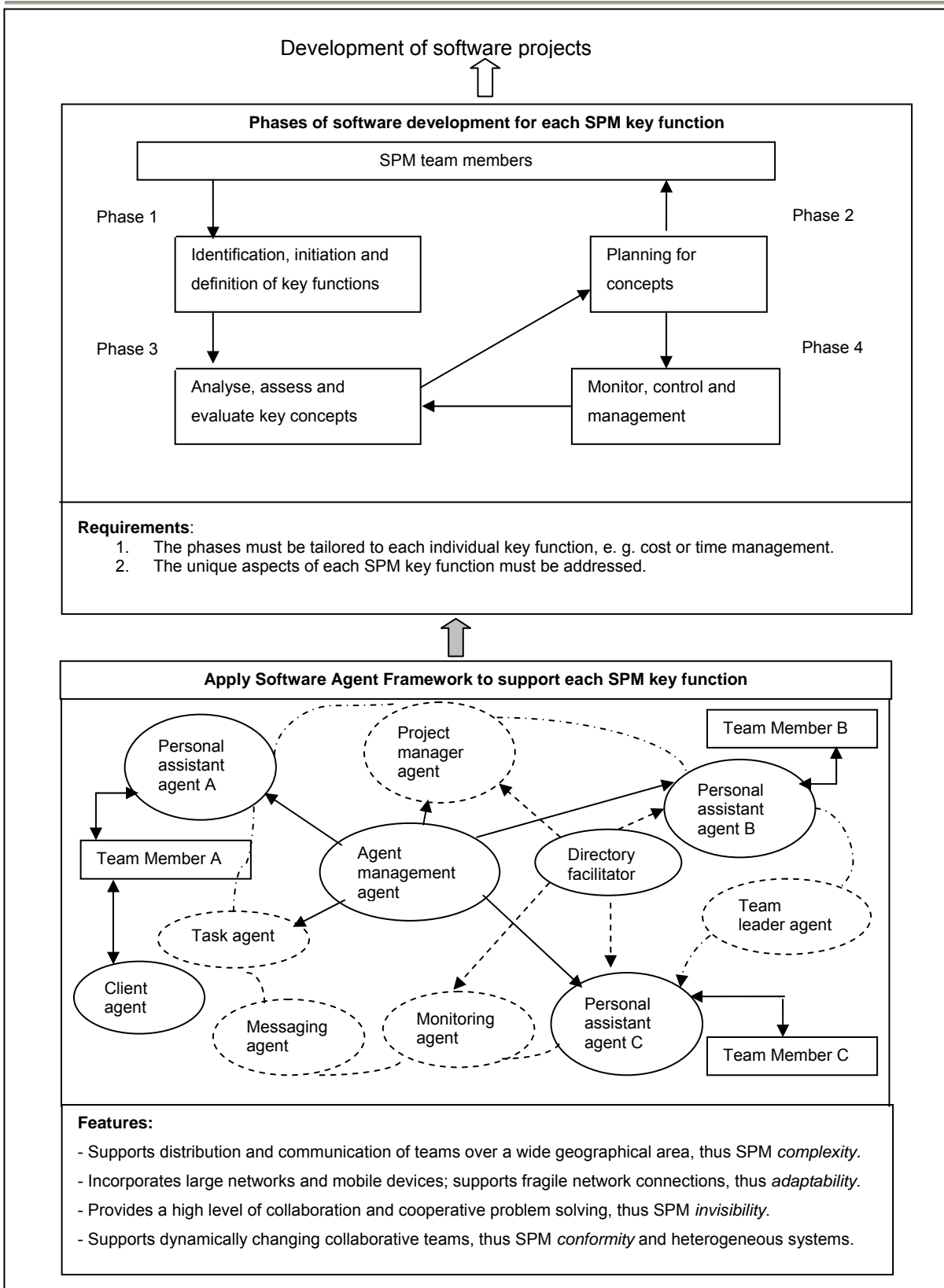


Figure 6.2 Conceptual view of the SPMSA model

6.3.1 Phases of software development for each SPM key function

Various methods and paradigms have been used to support the process of software development and as such software project management. The basic SPM processes were scrutinised in Chapter 3 (sections 3.6 and 3.7), and this resulted in a table depicting the basic phases of these processes, illustrated in Table 3.2. This table provides a summary of the basic phases of each function and as such constitutes the basis of the SPMSA model. On close inspection, overlapping phases can be identified as executed in each of these functions. An abstraction of these functions may be mapped to a generic model of software development that contains overlapping phases for each function (or process) of SPM. Thus the basic phases for each key function, illustrated in the conceptual model (top part) in Figure 6.2 (also referred to as a generic model of software development), are:

- Phase 1: Identify, initiate and define key functions.
- Phase 2: Plan the concepts of the particular process of the key function.
- Phase 3: Analyse, assess and evaluate key concepts of the key function concerned.
- Phase 4: Monitor, control and manage the functions of each key function.

The arrows indicate the order in which the phases are executed and follow on each other. Although a few additional tasks may exist depending on the specific key function concerned, all functions contain these basic phases. The requirements state that the implementation of these phases should be tailored to each individual key function, for example to cost or time management. In this way the unique aspects of each SPM key function will be addressed.

The upper section of Figure 6.2 therefore represents the SPM processes in the SPMSA model which will be supported by a software agent framework.

6.3.2 Software agent framework to support each SPM key function

Each of the key functions of SPM will be supported by a combination of one or more of the agents as indicated by the bottom half of Figure 6.2. The software agents will

support the generic functions for each of the key SPM processes (with minor practical differences, for example risk or time initiation).

Various types of agents or agent teams may be used to support the different phases of SPM. Chapter 4 contained a discussion of agents, and a distinction was made between software mobile and software stationary agents. To illustrate this basic configuration of agents supporting the SPM phases, a conceptual view of the operational environment of three team members (A, B and C) – which will probably be geographically dispersed – is depicted in Figure 6.2.

To describe how software agents can generically be employed to address different functions of SPM, a set of *agent teams* is used to address the functions and then to define specialised software agents operating within these teams (or on their own where applicable). The system is built regarding agents as components, which simplifies the design and programming of agents. The following specialised working mobile and stationary agents are used:

A *Personal Assistant agent (PA agent)* is used for each team member. This is an agent that supports each individual team member to accomplish his or her tasks by providing maximum assistance, as well as an interface between the team member and the other agents. This agent also has a collaborative nature and relies on other agents to provide it with the information that it needs to sustain its owner. The personal assistant agents are not computer-bound but human-bound, as their stakeholders may be required to work on different computers when working in a distributed environment.

The *Client agent* is a *stationary* agent responsible for a specialised task such as information retrieval or gathering. Client agents may or may not have intelligence, depending on their specific task, but they must have a collaborative nature to interact with other agents in their agent team.

The *Agent Management agent (AM)* is responsible for managing a team of agents and for ensuring coordination between the sub-tasks of the different members of a team to accomplish the objective of the agent team. This agent enables

communication, mobility, instantiation and destruction. The AM is central to communication and ensures that all messages arrive at their intended destinations. In performing this task, the AM must also track the distribution locations of agents with respect to their platforms and where mobile agents have moved to.

The *Task agent* is an agent that supports a specific project task. This agent collaborates with other objective and facilitator functions to support a specific task. This *mobile agent* is commonly invoked by a personal assistant agent to allow a stakeholder to work on a specific task, and is continuously monitored by a monitoring agent. An example can be that of an agent monitoring risk, schedules or aspects of time.

The *Monitoring agent* is responsible for monitoring tasks and for reporting back to different phases where scheduling and rescheduling of tasks, as well as the notification of stakeholders can take place. A monitoring agent is *mobile*, with intelligence, flexibility and strong collaborative properties.

The *Team Leader agent* is responsible for managing a team of agents and for ensuring coordination between the sub-tasks of the different members of a team to accomplish the objective of the agent team.

The *Project Manager (PM) agent* is an agent that takes on the project manager role, helps in the creation of the project, the initial specification of the tasks and allocation of tasks to personnel.

A *Messaging agent* is an agent responsible for carrying messages between different agent teams. This type of agent has strong collaborative characteristics and is by nature a mobile agent, since the different agent teams may work in a distributed environment.

Finally, the *Directory Facilitator (DF)* is an agent that provides a yellow pages functionality, which assists agents in discovering services provided by one another. This agent forms part of the facility provided by JADE, thus it is not included in the list of created agents in Figure 6.1.

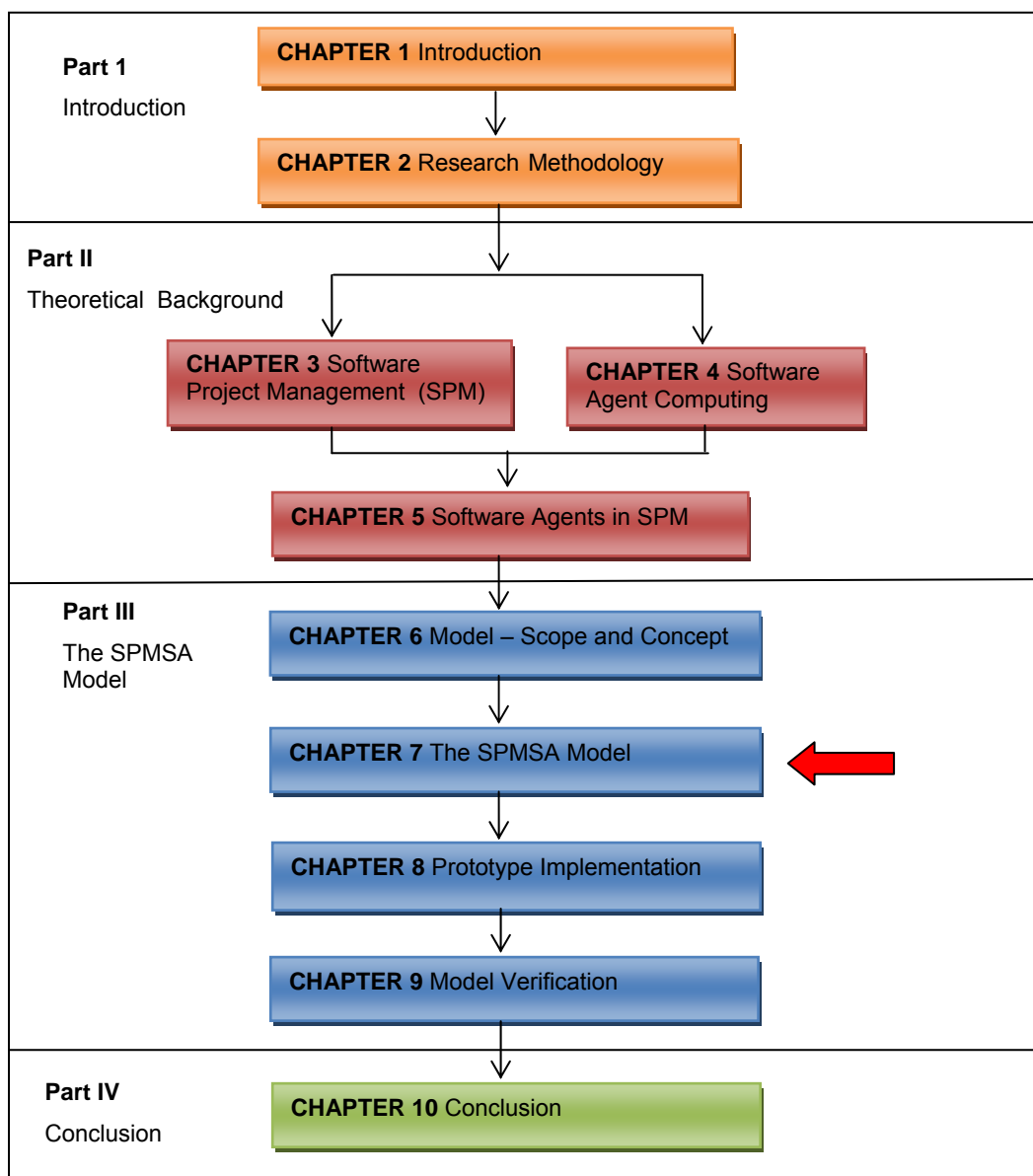
6.4 CONCLUSION

Chapter 6 provides a conceptual view of the scope and the basic concepts underlying the SPMSA model. It has been established that failure in the SPM area indicates that new paradigms should be used to support the specific requirements of this area. The salient features of agent technology imply that this paradigm will prove suitable to this need. The proposed model is thus specifically tailored to support the ever-changing environment and unique features of SPM.

Chapter 7 will regard each of the key areas of SPM and elaborate on each phase as illustrated in Figure 6.2. The aim is to compile a comprehensive model of SPM functionality to be supported by software agent technology. The phases depicted in Figure 6.2 will be scrutinised and applied to each of the eight key core and facilitating function areas of SPM, in order to compile the comprehensive SPMSA model.

CHAPTER 7

7 THE SPMSA MODEL



7.1 INTRODUCTION

In the previous chapter a conceptual view of the SPMSA model was provided by discussing the scope and concept thereof. The aim of the conceptual view was to place the SPM processes and the supporting software agent framework in context with regard to one another. The purpose of this chapter is to provide a detailed discussion of the SPMSA model. The phases of software development for each of the SPM key functions (as illustrated in Figure 6.2 in the previous chapter) will therefore be delineated and discussed in detail to compile the comprehensive SPMSA model.

The first part of the chapter is devoted to a discussion of the phases of each SPM key function, as well as a description of an agent team to support each SPM key function. A graphical representation will be compiled for each key function, which will consequently constitute the SPMSA model. The second part of the chapter will contain the full SPMSA model. The chapter will conclude with a table that depicts the advantages of using agents to address the unique and changing SPM environment.

7.2 SOFTWARE PROJECT MANAGEMENT KNOWLEDGE AREAS

Each of the distinct SPM knowledge areas introduced earlier in Chapter 3 is reconsidered briefly to delineate the phases of each key function to be supported by an agent framework. The steps comprising the processes of each of the key functions will be elaborated on in order to compile the SPMSA model that will enhance *all* of the SPM key functions involved.

In Chapter 3 a framework of the key functions in the entire SPM development arena was presented (see Figure 3.1). This framework contains the core and facilitating functions of software project management as key functions. These functions have been discussed in detail in Chapter 3. In order to compile a model that supports *all* of the SPM key functions involved Table 3.2 was compiled by the researcher as a

summary of the phases of the key functions that form the basis of the SPMSA model (Nienaber and Barnard, 2007). It is therefore inserted here again in order to be reviewed.

Table 3.2 Phases of the knowledge areas of SPM

Scope Management	Time Management	Cost Management	Quality Management	HR Management	Communication Management	Risk Management	Procurement Management
Initiation	Activity definition				Identification and planning	Risk identification	Procurement planning
Planning	Activity sequencing; Duration estimation	Resource planning	Planning	Organisational planning	Team support	Risk analysis and prioritisation	Solicitation planning
Definition	Time schedule development	Cost estimation	Assurance	Team development and staff acquisition	Information distribution	Risk Management planning	Solicitation and source selection
Verification	Time schedule control	Cost budgeting	Control	Management: Monitor and control	Performance reporting	Monitor	Contract administration
Change Control		Monitor, control			Administrative closure	Resolution	Contract closure

Table 3.2 illustrates the correlating phases of the core and facilitating functions of SPM as included in the SPMSA model. Integration management is considered as underlying part of these functions and not as separate knowledge area as in PMBOK (2004). The aim of the section that follows is to compile a graphical representation for each of the key functions, indicating how the key function may be supported by software agent technology in the SPMSA model. An abstraction of Table 3.2 is used when compiling these representations.

7.2.1 Scope management

The scope management function comprises the following specific phases: initiation, scope planning, scope definition, scope verification and scope change control. The purpose of each of these phases was highlighted in Chapter 3 and will therefore be mentioned only briefly in this section.

Figure 7.1 illustrates the different phases of the scope management function and the way in which agent teams cooperate to accomplish the objectives of these phases. The arrows indicate the flow of interaction, whether of communication or information, mostly via agents – as will be explained.

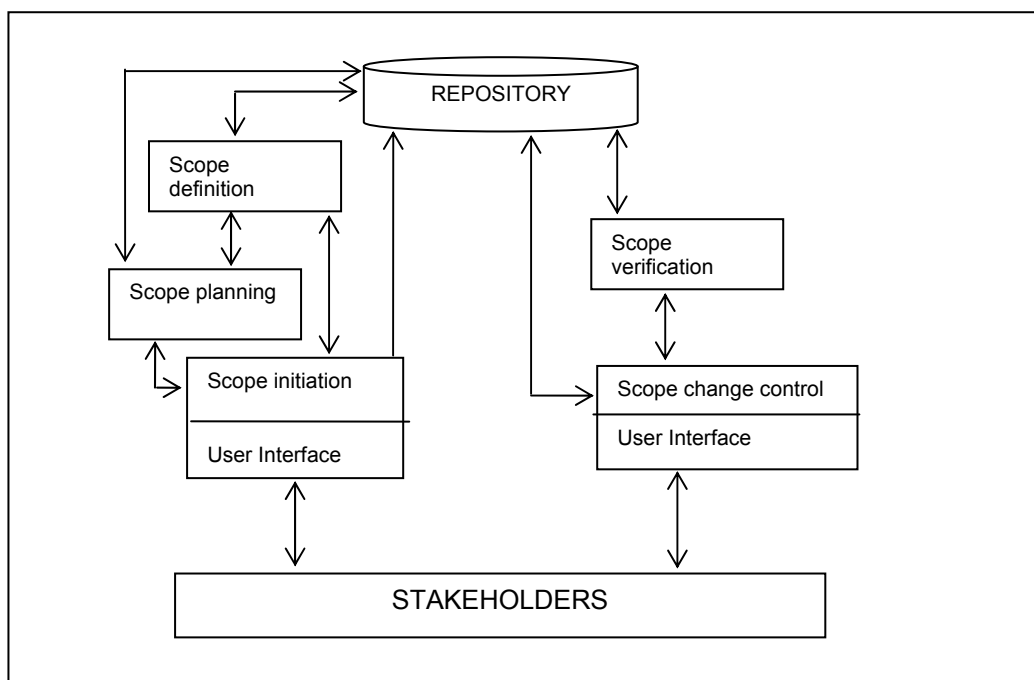


Figure 7.1 Scope management function

The software project manager, team members or other designated stakeholders interact with the *scope initiation phase* through a special user interface. The scope initiation phase involves committing an organisation to begin a project or continue a next phase of a project (Schwalbe, 2006). The output of this is a project charter, which will be stored in the repository. An organisation's strategic plan should guide

this project selection process. An important criterion for investing in IT projects includes supporting strategic business indicators, such as benefits and customer satisfaction, as well as financial incentives, such as a good rate of internal return (IRR), or net present value (NPV). The user interface is situated on top of the scope initiation phase and uses personal assistant agents, a project manager agent, task agents, an agent management agent and messaging agents. The directory facilitator agent assigns a personal assistant agent to the project manager that has supervision rights over other personal assistant agents. During scope initiation, the project manager defines team members that are assigned to this project. The project manager agent assigns a personal assistant agent to each team member, to be invoked with a user name and password. (For simplicity's sake, the username and password could be the same as a person's network login ID and password, but the choice depends on the individual, or the manager, should he or she decide differently for the sake of security). The personal assistant agent will support the team member it is associated with to automate various tasks such as the following: adding relevant documents to each task; sending these documents to all team members; sifting and organising email in priorities; communicating with other personal assistant agents concerning meeting scheduling; attaching agendas and distributing information to all committed team members. The information will be stored in the repository, from where it will be extracted when necessary. All agents will be managed by the agent management agent, also referred to as the agent management system. The agent management system is discussed in chapter 8 (8.3.1.1). Thus the task agents will support the project manager and each team member by automatically performing net present value analysis, return on investment and payback analysis on each proposed project, and so assist in the selection of a project. After completion, the project charter will be distributed to all concerned members as well as stored in the repository by the agent team. The scope initiation phase communicates with the scope definition phase and the scope planning phase.

Scope planning involves the development of documentation to provide the basis for future project decisions and provides guidance on other scope management processes. The project charter, project assumptions and project constraints are all inputs obtained from either the scope initiation phase or the repository, whereas the scope statement and project scope management plan are output to the system and placed in the repository. The scope-planning phase refines the scope and compiles a formal scope-planning document that contains all the above documents. The agent team will take this document to all relevant team members, and store it in the repository.

During *scope definition*, the team members define all tasks and deliverables of the project with input from the scope initiation and planning function, thereby creating a work breakdown structure (WBS). Client and task agents are available to automate standard tasks, for example to support the compilation of the WBS, do resource allocation, and to transport the formal scope definition document to all team members, requiring feedback from each. The document will be stored in the repository.

Scope verification involves formal acceptance of the project scope from all stakeholders. Input is obtained from the documents in the repository. Once the team members have accepted and verified the formal scope document during the scope verification phase, messaging agents store the completed document in the central repository.

The *scope change control phase* involves the control of changes to the project scope, and the use of an agent team that consists of messaging agents, task agents, client agents and a project manager agent. If changes are identified, this team evaluates and then implements the changes to the scope documents and resulting documents. The agent team uses task agents to gather information from the incoming messaging agents and client agents to perform scope integration and coordination. The messaging agents provide the documentation from the repository

and communicate with the verification phase, as well as the stakeholders, via the user interface. The directory facilitator provides a yellow page service to assist agents to discover services provided by other agents.

7.2.1.1 Advantages of software agent support for scope management

Current project management systems support individual aspects of the scope management function, such as a word processor to create documents and spreadsheet software to perform financial calculations. Project management software assists with the scheduling aspects of a project, but the scope and complexity of current software require facilities for coordinating independent activities and managing the project. They do not provide an integrated coordinated environment with set structure to support a uniform standard and methodology (O'Connor and Jenkins, 1999).

Agents excellently support the distributed retrieval and dissemination of information and documents. The agent system will automatically route workflow to all relevant team members, obtain and incorporate feedback into the system, and store documentation in the repository. The team members may not necessarily reside in the same area, and will probably be geographically dispersed. Some may even have access to the system only at certain times when visiting a site. Communication and coordination is enabled through the heterogeneous nature of an agent system. Agent systems can incorporate large network systems, as well as mobile devices. Tasks, such as financial analysis, can be embedded into mobile task agents, which can traverse the network and execute asynchronously and autonomously, without having to rely on a continued connection. Agent systems execute locally and thus reduce network load. Parallel execution also enables tasks to be executed at different workstations simultaneously. The team of agents can furthermore coordinate actions toward a similar goal, such as change control of the scope document.

7.2.2 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 the time that each task and activity will take, resulting in the overall duration of the project. The main processes of time management include activity definition, activity sequencing and activity duration estimation, time schedule development and time schedule control, as identified and discussed in Chapter 3.

A graphical representation of the different phases of the time management function and how agent teams cooperate to accomplish the objectives of these phases is illustrated in Figure 7.2.

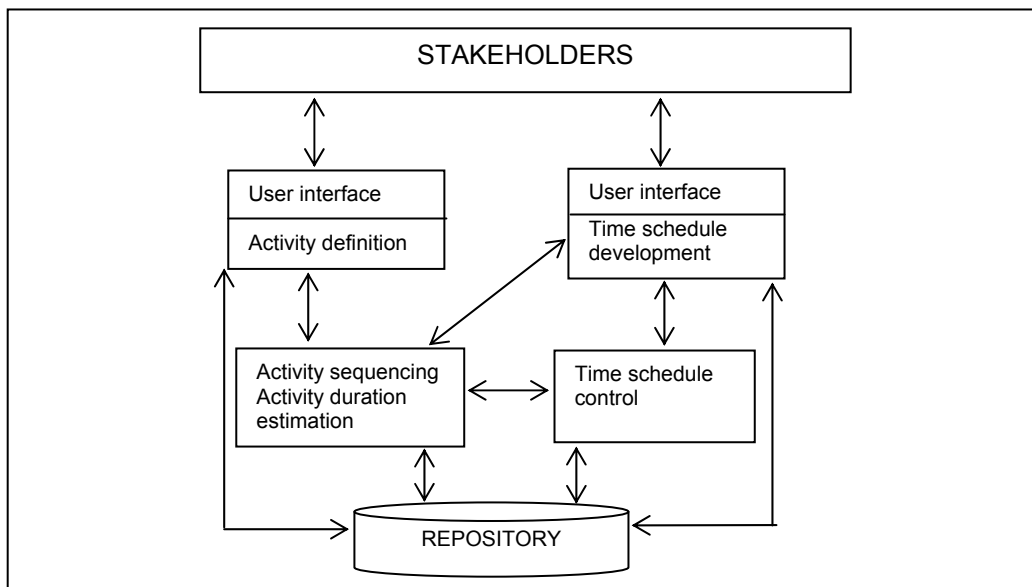


Figure 7.2 Time management function

The software project manager, team members or other designated stakeholders interact with the *activity definition phase* through a special user interface. During the activity definition phase each activity that must be performed to produce the project deliverables will be identified. Agent teams will support the team members in a similar way as in the scope management function. Each stakeholder and team member will be represented by a personal assistant agent, while a messaging agent will carry messages between agent teams and agents. Specific functions will be executed by task agents and client agents. The information will be obtained from

the WBS (compiled during scope management), the stakeholders and the scope definition document, which was stored in the repository.

The activities that have been defined during the activity definition phase will be input to the *activity sequencing* and *activity duration estimation phases*, also referred to as the *activity planning phase*. Activity sequencing entails identification of the relationships between project activities, while activity duration estimation entails estimating the time needed to complete activities. Software effort estimation techniques such as algorithmic models, analogy, Parkinson's estimations and parametric models can be programmed to be executed by task agents. In other words, function point analysis (Albrecht's and Mark II) and COCOMO, a parametric model, can both be programmed to be executed by a task agent (Benfield, Hendrickson and Galanti, 2006). The input will be provided by the activity definition phase and output will be sent to the time schedule development and time schedule control phases, as well as be stored in the repository. The task agents will support the duration estimation and activity sequencing phases.

The activity definition, sequencing and activity duration estimation phases constitute the basis for creating a project schedule. *Time schedule development* involves developing a schedule that considers the activity sequences, activity durations and resource requirements to complete the tasks and the project. Various standard SPM diagramming methods with supporting software are available (Schwalbe, 2006). These applications may be identified by personal assistant agents and suggested for use, such as network diagrams, i.e. activity-on-node and precedence networks, as well as Gantt charts, PERT techniques and critical path analysis. Calculations and simulations, such as "What if" analysis, can also be programmed to be executed by task and client agents, which will take the functionality to the team member and team leader's work area. However, the time schedule development phase will need interaction with the stakeholders (via the user interface) and team members to ensure commitment of all said members. By using input from the team members, previous experience will automatically be included in the calculations.

These diagrams will all be stored in the repository and they will be sent to the time schedule control phase.

The *time schedule control phase* uses an agent team that consists of messaging agents, monitoring agents, task agents and a project manager agent. This phase involves controlling and managing changes to the schedule. The project schedule is used as input to this phase and all changes are stored in the repository. If changes are identified, the agent team evaluates and then implements the changes to the project schedule and resulting documents. It uses task agents to gather information from the incoming messaging agents and client agents to perform schedule integration and coordination. The monitoring agents monitor all activities, while messaging agents circulate the documentation from the repository and communicate with all team members' personal assistant agents and the stakeholders. Milestones must form an integral part of this schedule, to enable and support the control phase.

7.2.2.1 Advantages of agent support for time management

The process of project planning and time management is notoriously inaccurate, mostly due to changing circumstances, additions or new information (Joslin and Poole, 2005). In response to these dynamic changes, time estimates may change, resources be reassigned and tasks may be changed. Thus, a dynamic planning system that can automatically incorporate changes will greatly benefit the project management environment. Although the use of artificial intelligence falls outside the scope of this thesis, the incorporation of agent-based simulation can be used for project time planning (Myers, Berry, Blythe, Conley, Gervasio, McGuinness, Morley, Pfeffer, Pollack and Tambe, 2007).

Schwalbe (2006) sites the times scheduling phase of the time management function as the main reason for conflict during the middle and end phases of a project. This conflict could be minimised by getting input from the team members and stakeholders during the activity definition and activity duration estimation phases. The agent system will continuously prompt users for input and interaction, thus

supporting communication and integration. Realistic time values will be derived which will benefit project professionals by minimizing the tendency to be overly optimistic. The agent system can in this way force continuous and realistic interaction between users and stakeholders.

Standard SPM software can do the basic diagrams, but the project manager/team members must be knowledgeable about these, i.e. if a manager does not know how to establish dependencies between tasks in Microsoft Project, it will result in errors. The agent system can support the team and the project manager to a larger extent as it will automate many tasks and will require only specific input.

7.2.3 Cost management

Cost management involves the managing of all financial aspects of a project to ensure that a project team completes a project within the approved budget. This includes resource planning, cost estimation, cost budgeting, and cost control and monitoring – as highlighted in Chapter 3. Figure 7.3 illustrates the different phases of the cost management function and how agent teams cooperate to accomplish the objectives of these phases.

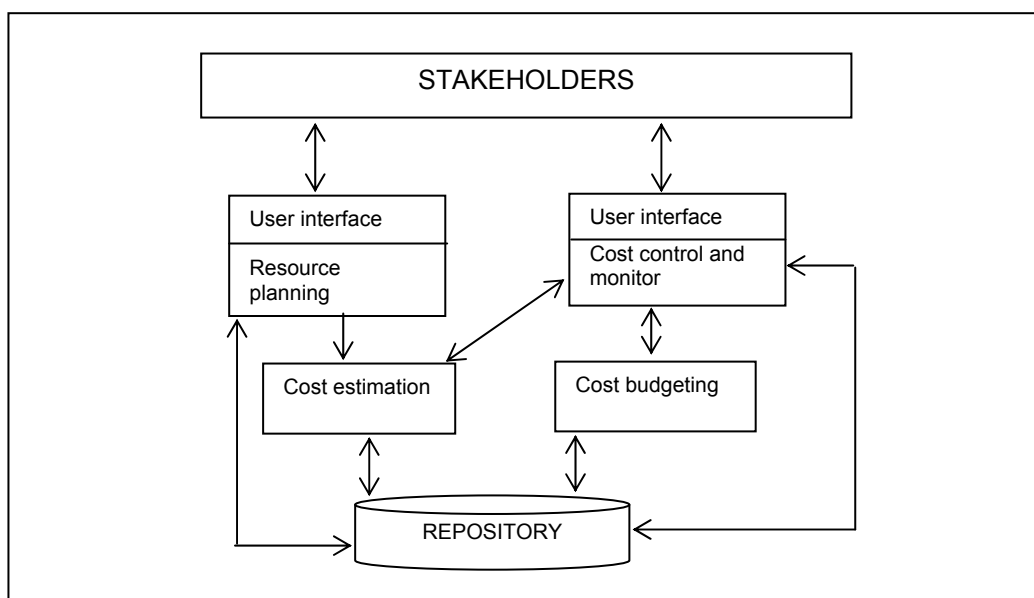


Figure 7.3 Cost management function

Although resource planning is not explicitly included by PMI (2004) as part of cost management, it is included as a function of the agent model to estimate project cost. During the *resource planning phase*, all resources and quantities of resources needed for a project are to be identified, and the output is a list of resources. Agent teams will support the phases in a similar way as in the scope and time management functions. Input concerning available resources is provided by the team leader and team members. The user interface, which is situated on top of the resource planning phase is used for interaction with all stakeholders. The resource planning phase uses personal assistant agents, task agents and messaging agents. A personal assistant agent will support each individual team member to accomplish his or her tasks by providing maximum assistance, as well as by providing an interface between the team member and the other agents. The output of this function will be communicated to the cost estimation function as well as stored in the repository. The resource planning will be supported by the directory facilitator to select any pre-existing functions encapsulated in agents.

The *cost estimation phase* involves developing an estimate of the cost of resources needed for this project. Input to this function will be information from the messaging agents of the resource planning function, and the functioning will be supported by messaging, task and client agents. Task and client agents will automate financial calculations such as rough order of magnitude (ROM) estimates, return on investment estimates (ROI), budgetary estimates, derived estimates (Benfield et al., 2006). These estimations will constitute a cost management plan. Various computerised tools are available and can be integrated to support this action, such as analogous estimates, bottom-up costing and parametric modelling (Hughes and Cotterell, 2006). The cost estimation phase interacts with the resource planning phase, the cost control and monitor phase and the repository where the estimation plan will be stored.

The *cost budgeting phase* entails allocating the project cost estimates to specific items. Input to this is the WBS and project schedule, which are obtained from the

repository and used as input to the cost control function. Agent teams will be used to monitor the cost and automatically update expenditure. All changes will be stored in the repository.

The cost control and monitor phase involves the monitoring of cost performance and will be supported by a monitoring agent to monitor tasks and compare expenditure to budgeted amounts. This function will interact with the cost budgeting and cost estimation phases, storing all relevant documents, for instance the estimation plan, in the repository. Notification of stakeholders can take place via the user interface. A client agent will be responsible for specialised tasks, while the team leader agent will be responsible for managing the team of agents.

7.2.3.1 Advantages of agent support for cost management

Current approaches rely on the team leader or identified team members to manually or electronically update and maintain cost expenditure (Hughes and Cotterell, 2006). Current PM approaches provide capabilities for financial calculations such as ROI, NPV, etc., but the project leader or member still has to provide the information. Agent teams can automate this on specified times, as part of preparing the budget thus eliminating the 'human error' aspect. Calculations can be executed by agent teams and changes can be incorporated dynamically. Benfield et al. (2006) report a developer productivity of 350% for systems supported with a type of agent support. The agent approach also reduces complexity, incorporates changes easily and thus compiles solutions faster than traditional programming paradigms.

7.2.4 Quality management

Project quality management involves all activities of the overall management function that determine the quality policy, objectives and responsibility. It implements these by means of quality planning, quality assurance and quality control within the quality system. Each of these processes has been discussed in Chapter 3.

Figure 7.4 illustrates the different phases in the quality management function and how agent teams cooperate to accomplish the objectives of these phases.

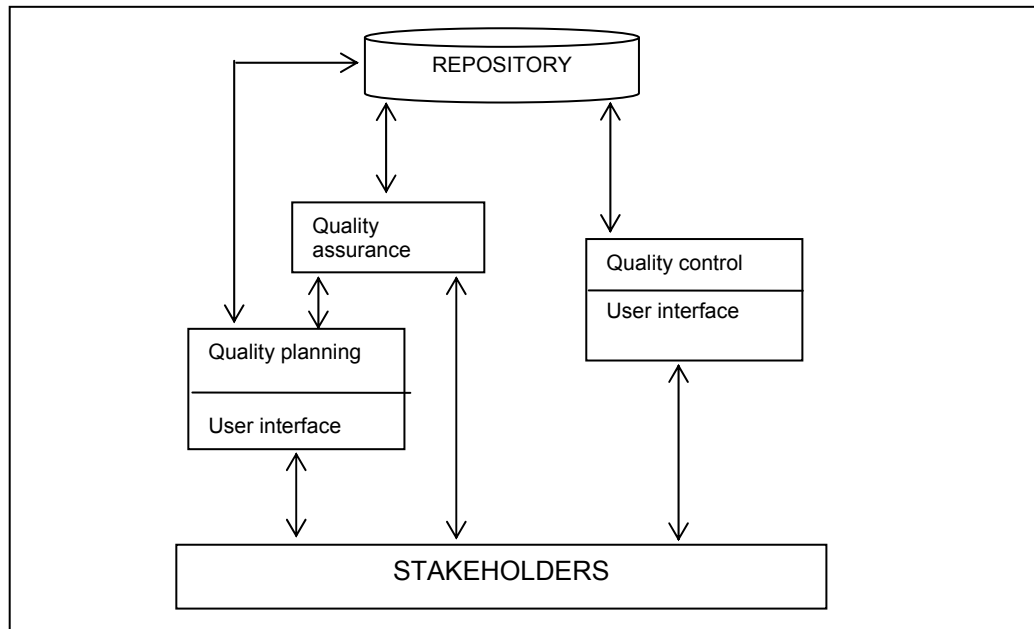


Figure 7.4 Quality management function

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

During the *quality planning phase*, interaction with the user interface enables communication between stakeholders and the agent team. Quality standards are identified and quality measures set. A plan devised to adhere to these standards will

result in the quality plan, which will be stored in the repository for further referencing. Agents utilised are personal assistant agents to assist each team member, task agents to set and identify relevant quality measures, messaging agents to communicate to stakeholders and teams, monitoring agents to receive and distribute documents, and team management agents to coordinate agents.

Quality assurance involves evaluating overall performance regularly to ensure conformance to the set standards. Task agents will support this evaluation to ensure compliance to set standards, traverse the network of team members on a regular basis and, when problems are encountered, give warning messages to personal assistant agents who will communicate with individual team members, as well as with the stakeholders. Input will be from the quality planning phase, namely the quality plan. Quality audits or reviews are used to support this function and personal assistant agents will be responsible for setting schedules, agendas and meetings; for distributing information needed, and for instructing messaging agents to deliver messages, all of which will be stored in the repository.

The team of agents will support the *quality control phase* while monitoring the activities and products of the project to ensure compliance with the standards. As quality control involves acceptance of the work in hand, the user interface of this phase ensures communication between stakeholders on a regular basis. Task agents will traverse the network regularly to monitor tasks and report back to all stakeholders. Various project management techniques may be used, such as pareto analysis, statistical sampling and quality control charts, which can be programmed and executed by agents (Olson, 2004). A library of tools and techniques may be made available and best choices may be selected by agents to support each team member. If any rework or process adjustment is necessary, it will be communicated and coordinated between task and personal assistant agents. Monitoring agents will control and check that rework and adjustments are executed and that change control documentation is stored in the repository. Monitoring agents will also monitor all agent activity.

7.2.4.1 Advantages of agent support for quality management

There is a need for improvement on this level of SPM. Hughes and Cotterell (2006) recommend that quality aspects of the project plan be reviewed on an ongoing basis. Traditionally, project quality control depends primarily on either the project leader or a specific allocated team, thus on human input (Schwalbe, 2006). Software agent teams may assist in these endeavours by continually prompting team members for input, regularly measuring workflow and rework status.

7.2.5 Human resource management

Human resource management includes all the people concerned with a project. These will include the project stakeholders, sponsors, customers, project team members, support staff, suppliers supporting the project, as well as any other person or item needed to complete the project. The main focus of this process is to allocate resources to activities, and to create a work schedule utilising these resources from the activity plan. Human resource management consist of the following phases, namely the organisational planning phase; the HR team development and staff acquisition phase; and the monitor and control (management) phase.

Figure 7.5 illustrates the different phases in the human resource management function and how agent teams cooperate to accomplish the objectives of these phases.

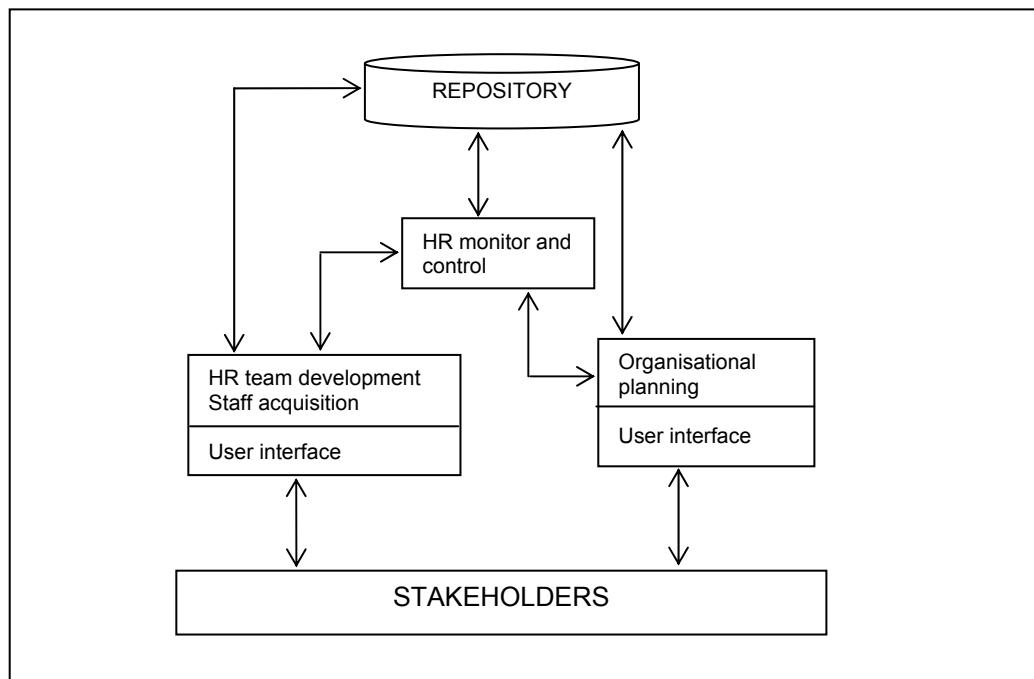


Figure 7.5 Human resource management function

The software project manager or other designated stakeholders interact with the *organisational planning phase* through a special user interface. This user interface, which sits on top of the organisational planning function, uses personal assistant agents, task agents, messaging agents and a project manager agent. The Directory facilitator agent assigns a personal assistant agent to the project manager, which has supervision rights over other personal assistant agents. The project manager agent assigns a personal assistant agent to each team member. Once the user has entered the required information into the system via the user interface, messaging agents take the information to a central repository and to the HR monitor and control phase. The organisational planning phase involves identifying and documenting project roles, responsibilities and relationships. A staffing management plan and organisational chart will result and both will be stored in the central repository.

The *HR team development and staff acquisition phase* entails staff acquisition or assigning the needed personnel to the project, as well as building individual and

group skills needed to enhance the project. This will primarily be executed by the project manager, with a team assisting him/her. The project manager or other designated stakeholders interact with the team development and staff acquisition phase through a user interface. A team of agents similar to the team supporting the previous functions will support this phase. Thus, the personal assistant agent will support the project leader in identifying personnel with the needed skills and in compiling a suitable team. Task agents will match the existing skills and knowledge of staff to the required skills and knowledge, in order to identify areas to be developed. Agents involved will be task agents for specific tasks, messaging agents to communicate and deliver messages, and monitoring agents to control and check. The HR team development and staff acquisition phase interacts with the user interface, the HR monitoring and control phase, and the repository by storing finalised documents.

The *HR monitor and control phase* includes tracking team performance, providing timely feedback, resolving issues and conflict, as well as coordinating changes. The primary responsibilities of the client and task agents are to facilitate teamwork, perform scheduling on teamwork, and distribute collaborative documents. The HR monitor and control function interacts between the organisational planning phase and the team development and staff acquisition phase. Task agents will execute monitoring tasks by continuously traversing the network of team members and reporting on the status of all tasks and subtasks. Messaging agents will receive and carry information, and the team leader agent will manage the team of agents. These agents will interact and regulate the functioning of the HR monitor and control function. Completed and intermediate documentation will be stored in the repository.

7.2.5.1 Advantages of agent support for HR management

Current systems contain software that may be used for support to facilitate organisational planning and assign resources, but people skills will be the primary objective of this phase. Current approaches support the project leader and team only with loose software support, such as responsibility assignment matrixes,

resource histograms and other reporting features (Hughes and Cotterell, 2006). The framework of agents will support the team with an integrated approach of support, mainly through the personal assistant agent and task agents. Coordination and communication will also be supported.

7.2.6 Communication management

Communications management creates an environment for interaction and ensures timely and appropriate generation, collection, dissemination, storage and disposition of project information. This function consists of five distinct phases, namely communications identification and planning, team support, information distribution, performance reporting and administrative closure. These have been highlighted and explained in Chapter 3.

Figure 7.6 illustrates the different phases in the communications management function and how agent teams cooperate to accomplish the objectives of these phases.

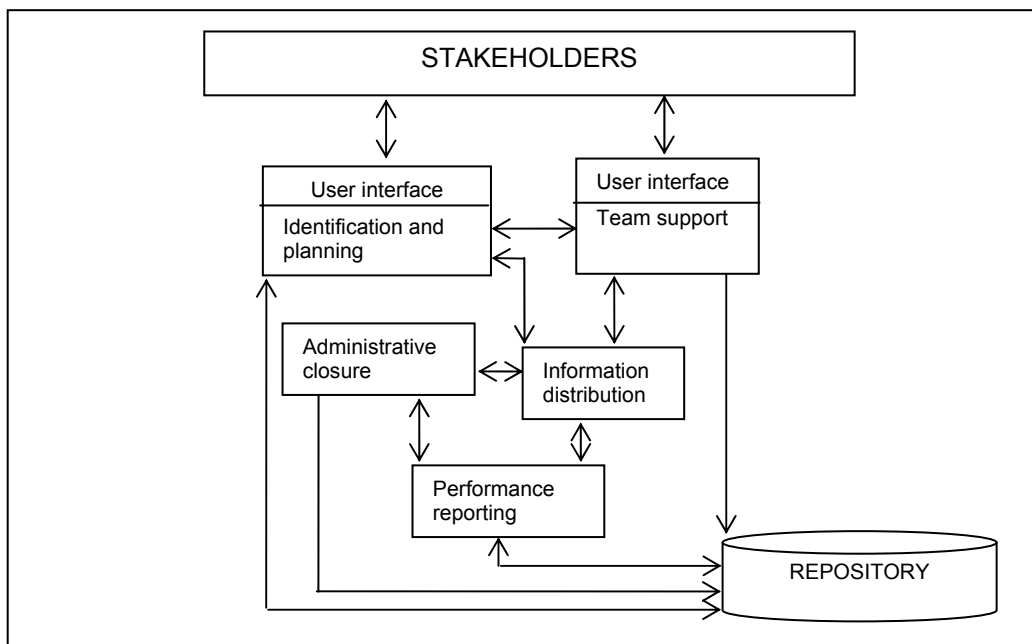


Figure 7.6 Communication management function

The software project manager or other designated stakeholders interact with the *identification and planning phase* through a special user interface. This user interface, which sits on top of the communications identification and planning function, uses personal assistant agents, task agents and messaging agents. During interaction with the interface, the user defines team members or relevant stakeholders and the tasks that are assigned to them, and defines milestones, objectives, etc. The project manager agent uses the directory facilitator to assign a personal assistant agent to the project manager, as well as to each team member. This agent may be invoked with a user name and password. Once the user has entered the required information into the system, messaging agents take the information to a central repository and to the information distribution function.

The *information distribution phase* uses an agent team that consists of messaging agents, task agents, client agents and a team leader agent. The agent team of this function accepts incoming messaging agents from the user interface of the team support phase and uses its own messaging agents to interact with the identification and planning phase, team support phase, performance reporting phase and the administrative closure phase. It also uses client agents to gather information from the incoming messaging agents and task agents to perform information integration and coordination.

As before, task agents are included for specialised computing tasks. For the information distribution phase, task agents may or may not be included at this level, depending on how elaborative the client agents are. The researcher advocates the use of task agents to simplify the design and improve the maintenance of the SPM tool software. As mentioned before, the client agent typically has a number of functions that include interacting with (and thus receiving) incoming messaging agents; understanding (interpreting) incoming information; translating incoming information to a syntax that makes it possible to be processed; processing the incoming information; and deciding on distribution conduct (based on its generic approach to handling information as well as previous knowledge and experience).

The client agent is also tasked with the responsibility to interact with the outgoing messaging agents that must disseminate the processed information and send the information to the administrative closure phase. The latter interacts with the central repository. To simplify the design of a client agent, these individual tasks can be designed as task agents reporting to the client agent via the team leader agent.

The *team support phase* is primarily responsible for collaborative scheduling tasks. The phase prescribed by PMBOK (2004), namely *managing stakeholders* will be included in the team support phase of the communication management function. The software project manager or other designated stakeholders interact with the team support phase through a user interface. Agents associated with scheduling are monitoring agents, personal assistant agents, client agents (and task agents where applicable), as well as messaging agents. Messaging agents are defined as before. Monitoring agents are responsible for monitoring the incoming messages from messaging agents. They subsequently determine the necessity or urgency to suggest new or earlier meeting schedules than those already scheduled during the previous communication rounds, or by the team support phase. The primary responsibilities of the client and task agents are to facilitate teamwork, perform scheduling tasks on teamwork, and distribute collaborative documents. When an individual team member works on a collaborative document, his or her personal assistant agent must be cognisant of any extraordinary circumstances when the user falls behind schedule. This could for example be done by special-prompting-task-agents that ask specific questions or monitoring agents that compare set dates to real dates. The personal assistant agent passes this information to the (manager) monitoring agent, which either sends the agent to the general client agent at this level, or makes special suggestions with regard to extraordinary meetings to be scheduled. A user interface is available at this level, through which team members can interact with the collaborative task environment.

The *administrative closure phase* interacts between the information distribution phase, performance reporting phase and the central repository. It also keeps a

history by using monitoring agents to coordinate incoming reports before storing or archiving the information to the central repository. As expected, this function includes both messaging agents and client agents (potentially also task agents to assist the client agents) to coordinate the incoming reports and archiving processes.

The performance reporting phase entails the generation of reports such as status, progress and forecasting reports. The performance reporting phase interacts between the administrative closure phase, the information distribution phase and the central repository. It receives input from the administrative closure and information distribution phases through the use of monitoring agents to coordinate incoming reports before generating required reports and then archiving the information to the central repository. This function requires both messaging agents and task agents to coordinate the generation of reports and archiving processes.

7.2.6.1 Advantages of agent support for communication management

Traditional SPM tools use a passive reporting mechanism that does not provide sufficient reporting support to a collaborative distributed system (Chen et al., 2003). Several software tools are used, such as a word processor to create documents, spreadsheet software to perform financial calculations, as well as standard project management software that can assist in drawing diagrams. However, these are separate tools and do not provide an integrated, autonomous environment to support all phases. Communication is enhanced and supported by the use of an agent system that prompts the users for input and thus eliminates human type of errors, such as forgetting to document all details (leading to insufficient project documentation). Another common problem in communication is that many project processes, contexts, rationales, or artefacts may not be captured at all. An electronic repository supported by an agent system, will automate and monitor documentation storing and retrieving, that in turn will overcome these disadvantages (Nienaber and Cloete, 2003).

Distributed retrieval and dissemination of information and documents are also supported by the agent system. Workflow will automatically be routed to all relevant team members, feedback will be obtained, and documentation will be stored in the repository. The team members will probably be geographically dispersed, but communication and coordination will be enabled through the heterogeneous nature of an agent system.

7.2.7 Risk management

Various models or frameworks exist, which may be used to identify and address risk associated with software project development.

Based on Boehm's model (1991), software risk management consists of the following phases: risk assessment that includes risk identification, risk analysis and risk prioritisation, as well as risk control, comprising risk management planning, risk resolution, and risk monitoring. Figure 7.7 illustrates the different phases in the risk management function (as discussed in Chapter 3) and how agent teams cooperate to accomplish the objectives of these phases.

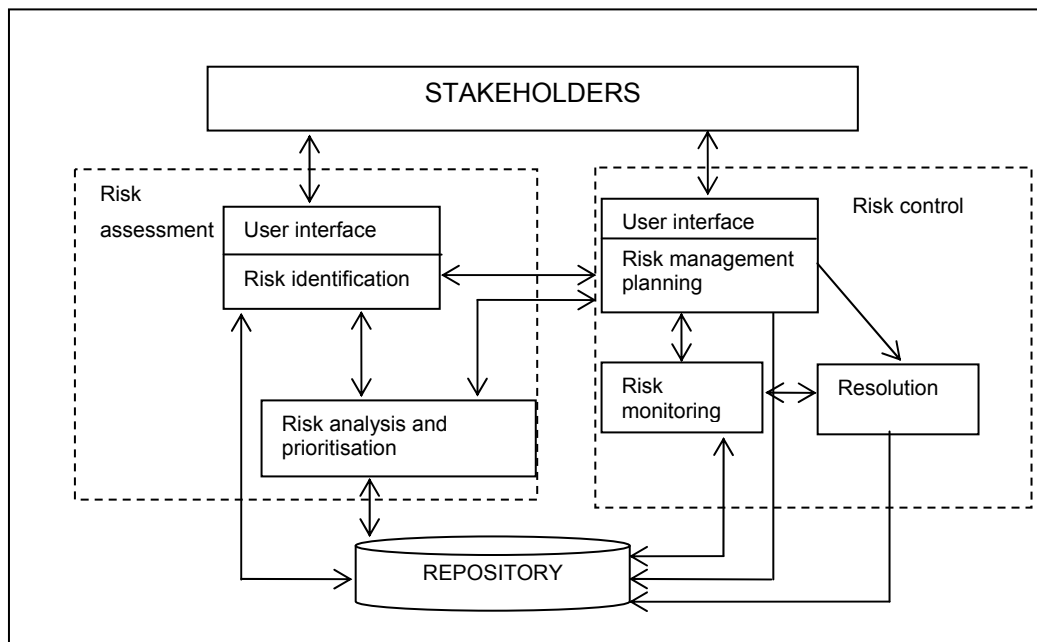


Figure 7.7 Risk management function

The *risk identification phase* entails the identification of specific risks by the project manager and the team members. The software project manager or other designated stakeholders interact with the risk identification phase through a special user interface. This user interface, situated on top of the risk identification phase, uses personal assistant agents, task agents and messaging agents. The project manager agent uses the directory facilitator agent to identify personal assistant agents of all the members of the team. The project manager then identifies possible risks for this project. The agent management agent is responsible for managing the team of agents, ensuring coordination between the sub-tasks, communication between agents and the location distribution of agents. The project leader defines the parameters of the project with the assistance of the personal assistant agents, adds tasks and subtasks, and allocates tasks to team members with the support of the personal assistant agents. Once the user has entered the required information into the system, messaging agents take the information to a central repository and to the risk analysis and prioritisation phase.

During the *risk analysis and prioritisation phase*, the task agent further traverses the distributed network of team members, communicating with each team member's personal assistant agent. Each team member will enter information concerning risk probability (e.g. identify on a scale of 1 to 100 the probability of a certain risk occurring). The task agent will also perform calculations regarding the impact on the assets, where known, if the risk occurs. Furthermore, each team member will be prompted for information concerning risk areas at regular time intervals. Thus, this function will enhance the risk management function by continuously updating team members on the probability of a certain risk occurring. The status of each task will also be monitored by the task and monitoring agent. All intermediate information will be communicated to the risk management planning phase, as well as be stored in the repository.

Risk management planning involves the developing of strategies to address risks, should they occur. A risk management plan that includes aspects such as risk

mitigation strategies for technical cost and schedule risks, as well as contingency plans, will be stored in the repository for further reference. Interaction with the stakeholders, all team members, as well as with the risk identification, risk analysis, risk prioritisation, risk-monitoring and risk resolution phases is necessary.

The *risk-monitoring phase* will provide the team and the team leader with information on the status of each specific task, e.g. a warning message if tasks or deliverables are overdue or on schedule; the probability of occurrence of identified risks. The monitoring agent will be responsible for monitoring tasks, reporting back to the PAs where rescheduling of tasks as well as the notification of stakeholders can take place. Task documents, attached to a specific task, will also be monitored. This phase will interact with the risk management planning phase, the risk resolution phase, as well as with the repository.

Risk resolution will be supported by the agent team. Client and task agents are available to automate standard tasks, for example to support the compilation of the risk resolution plan, to do simulations and benchmarks, and to transport all formal documentation such as the formal risk resolution planning document to all team members, requiring feedback from each. The document will be stored in the repository.

7.2.7.1 Advantages of agent support for risk management

Current risk management approaches consist of a variety of application packages that address various areas of functionality in the SPM area. These applications require pre-knowledge of the project manager and are time-consuming to use. The incorrect usage, or omission of certain items will result in errors in calculations and estimations. Input is also time-consuming and may be omitted in certain circumstances. The software agent approach will have the advantage that the SPM process is enhanced and supported by automated process input. Real-time progress measurement as sustained by continuous input will help to identify potential risks early and support the project manager to be proactive. The team

leader will be informed of the status of all tasks of the specific project. Thus interaction with the user, project manager and other stakeholders will be optimised.

The task agent will traverse the distributed network for input from all team members on the selected risks, which will enhance the process by sending the functionality to the various team members. Thus, communication overhead and network load are lessened. Task documents will also automatically be distributed over the distributed network, lessening the work load of each team member.

Distributed teams can also communicate and coordinate through this heterogeneous nature of the agent system.

7.2.8 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 entails acquiring services or goods from an outside source. Procurement management thus comprises the methods and procedures stipulated by an organisation to facilitate the acquisition of such products. Project procurement management consists of the following processes, namely procurement planning, solicitation planning, solicitation and source selection, contract administration, and contract closure as highlighted by the researcher in Chapter 3.

Figure 7.8 illustrates the different phases of the procurement management function and how agent teams cooperate to accomplish the objectives of these phases.

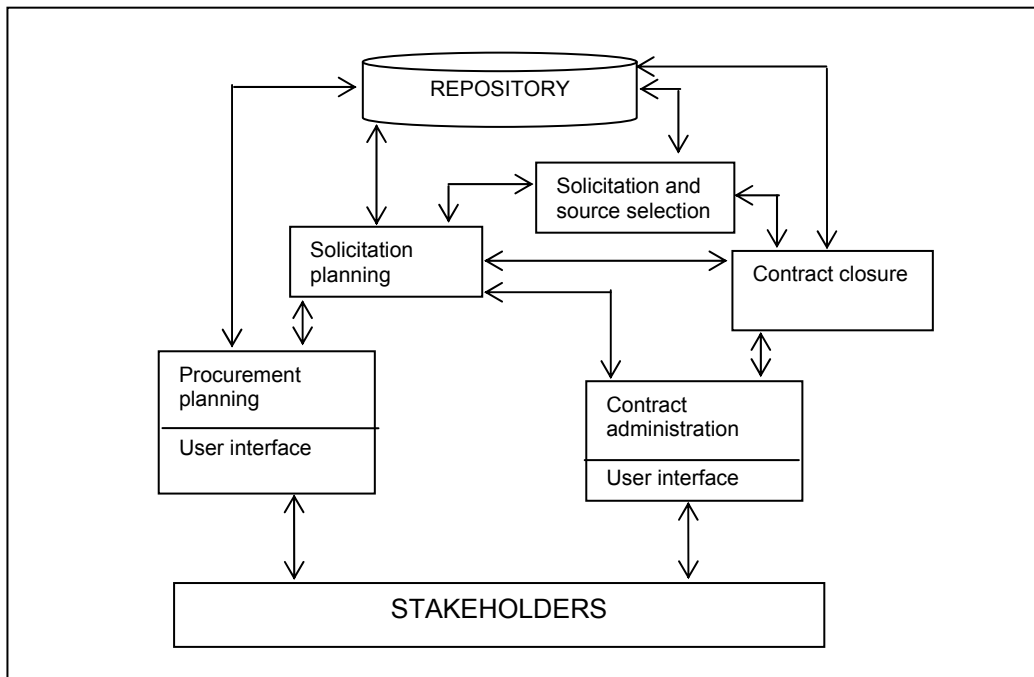


Figure 7.8 Procurement management function

The phases of this function will each be supported by agent teams that are similar to the agent teams in the previous sections.

The *procurement planning phase* involves identifying what to procure and when. These decisions will constitute the procurement management plan. Interaction is needed from the stakeholders and team members and will be enabled by the user interface. This user interface, situated on top of the procurement planning phase, uses personal assistant agents, task agents and messaging agents. The personal assistant agent of each member will support the member to identify and specify items needed for the task in hand. The messaging agent will take this document to all relevant team members, via the user interface, the solicitation planning phase and store the completed document in the repository.

During the *solicitation planning phase* product requirements are documented and resources identified. Procurement documents and rules are set. Client and task

agents are available to automate standard tasks, for example to support the compilation of the RFP (Request for Proposal), and to compile the evaluation criteria based on input from all team members. The procurement documents will be stored in the repository. This phase interacts with the solicitation and source selection phase, contract administration, as well as contract closure phase.

The *solicitation and source selection phase* entails the process of obtaining quotations, bids and offers, and selecting the most appropriate one. This phase is supported by the team of agents. The personal assistant agent supports each team member with individual tasks, while task agents will do specific comparisons. Several studies have been completed on negotiating agents (Kephart and Chess, 2003; Chmiel, Czech, and Paprzycki, 2004a). This process can thus be fully supported by a team of bidding and negotiating agents. All documents, such as quotations, bids and offers, will be stored in the repository, as well as be sent to the contract closure phase.

The *contract administration phase* refers to the management of the contract in accordance with outside stakeholders, such as suppliers. As this phase entails interaction with the stakeholders and suppliers, such interaction is enabled by the user interface. The contract administration phase will again be supported by an agent team, supporting the team member/s who execute this function via the user interface. Contract administration detail will be communicated with the contract closure phase.

Contract closure involves a procurement audit, as well as formal acceptance and closure of the contract. Input is obtained from the solicitation and source selection phase, contract administration phase, and solicitation documents in the repository. Once the team members have accepted and verified the product concerned in the contract, messaging agents store the completed document in the central repository.

7.2.8.1 Advantages of agent support for procurement management

Current systems rely on individual team members to execute the procurement acquisition process. The procurement management function can be extensively supported by bidding and negotiating agents. Research has been done and is still being done on automated bidding agents that can bid for services or products in electronic markets, without direct human intervention (Kephart and Chess, 2003, Badica, Popescu, Vukmirovic, Gawinecki, Kobzdej, Ganzha and Paprzycki, 2008). Autonomous agents consisting of bidding, buyer, seller and middle agents may bid for products or services. These agents are able to negotiate according to inbuilt negotiation algorithms and conclude transactions to maximise gain for an organisation (Shehory, Goldstein, Shulman, Sturm and Yurovitsky, 2002). Buyer agents are sometimes referred to as shopping agents (Kephart and Chess, 2003). Pricing agents may also provide team members with information concerning prices in the market (Kephart and Greenwald, 1999). This will free the development team and management team from having to deal with the actions of acquiring information on products or services, bidding to buy them, negotiating and concluding the transaction – thus allowing them more time for quality product development.

7.3 THE “SOFTWARE PROJECT MANAGEMENT SUPPORTED BY SOFTWARE AGENTS” (SPMSA) MODEL

The proposed SPMSA model is unique as it supports *each* key function of SPM with a *team of software agents*. In the previous section it was illustrated how an agent team supports each of the SPM key functions. Table 7.1 provides a summary of the purpose of each agent that forms part of the agent teams of the different SPM key functions of the SPMSA model.

Table 7.1 Agents and their tasks

Agents	Purpose	Mobile	Stationary
Agent management agent	<ul style="list-style-type: none"> • Manages the team of agents • Keeps track of the distribution location of all agents • Enables communication of agents • Enables mobility of agents • Tracks instantiation of tasks 		X
Client agent	<ul style="list-style-type: none"> • Executes a specialised task at a workstation • Interacts with the agent team • Receives input from task agent 		X
Directory Facilitator	<ul style="list-style-type: none"> • Automated JADE facility • Provides yellow pages functionality to agents • Provides agents with information on services provided by other agents 		X
Personal assistant agent	<ul style="list-style-type: none"> • Allocated to a specific team member • Assists the team member • Interface between team member and other agents • Collaborative nature 		X
Messaging agent	<ul style="list-style-type: none"> • Traverses the network of agents • Carries messages to and from agents • Collaborates between agents 	X	
Monitoring agent	<ul style="list-style-type: none"> • Monitors agent movement • Monitors tasks and activities 	X	

Agents	Purpose	Mobile	Stationary
	<ul style="list-style-type: none"> Coordinates agents 		
Project manager agent	<ul style="list-style-type: none"> Supports and directs the team of agents Takes on project manager role Helps create and initialise project Specifies tasks Allocates tasks to members 	X	
Task agent	<ul style="list-style-type: none"> Supports a specific task, e. g. information gathering, information distribution, information retrieval Traverses the network of team members for input Calculates various measures, such as probability of risk occurring, ROI, NPV Gives feedback to personal assistant agents 	X	
Team leader agent	<ul style="list-style-type: none"> Manages the team of agents 	X	

The graphical representations compiled for each of the SPM key functions (as depicted in Figures 7.1 to 7.8) are mapped to form the SPMSA model. The core functions and the facilitating functions are presented in Figure 7.9 and Figure 7.10 respectively. These two figures represent the entire SPMSA model, but due to space constraints they are spread over two pages.

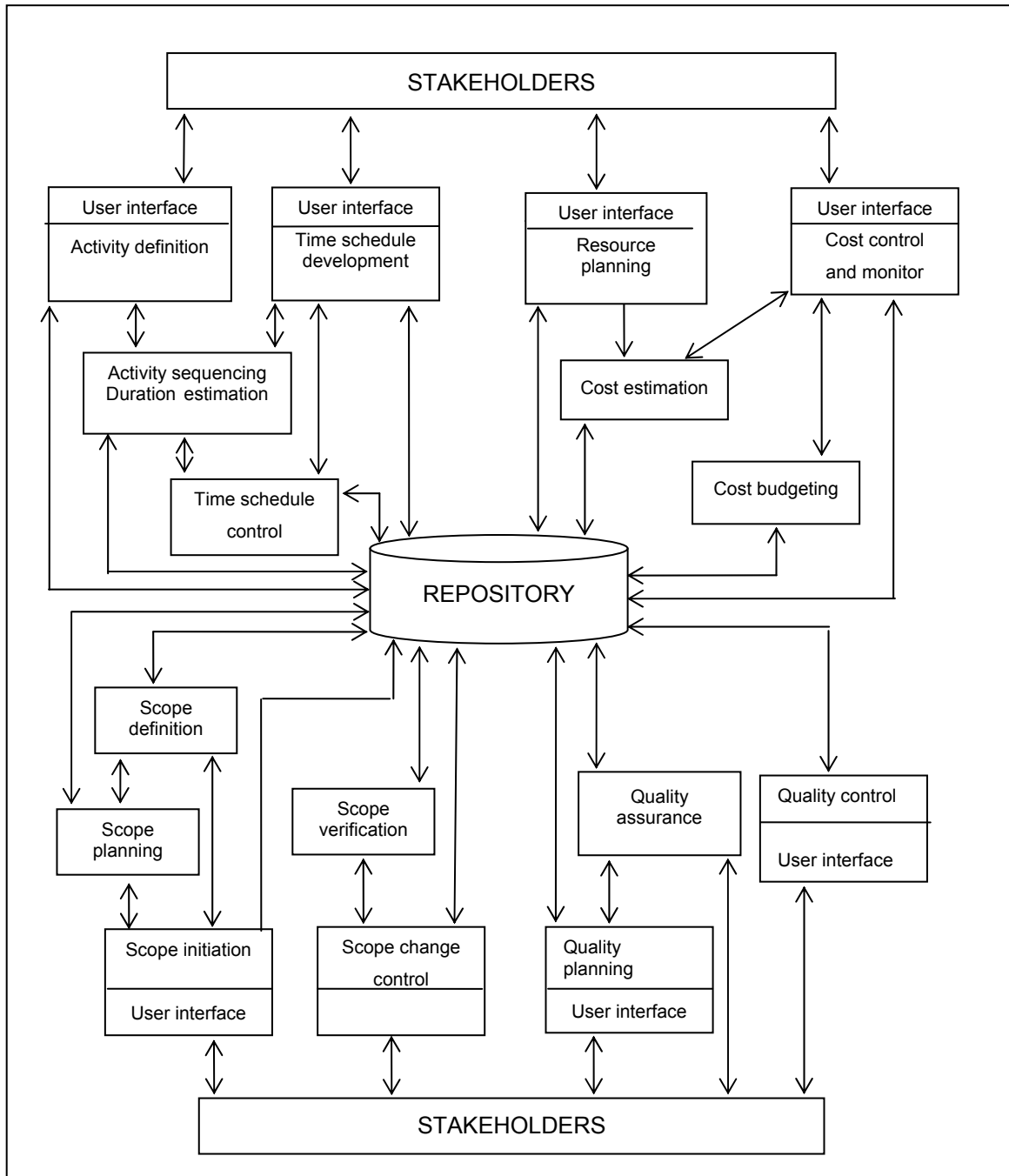


Figure 7.9 The SPMSA model - core functions

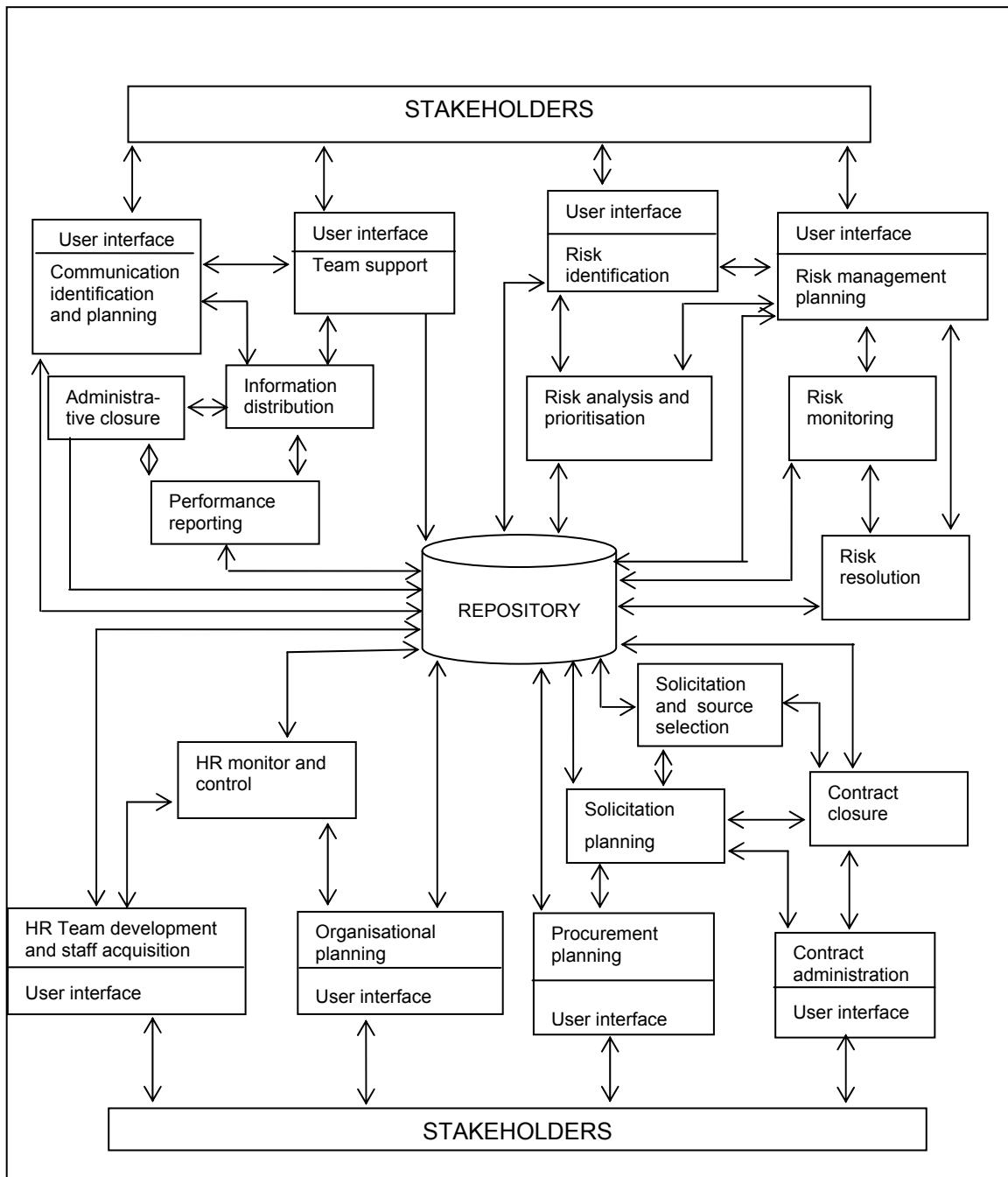


Figure 7.10 The SPMSA model - facilitating functions

The SPMSA model is unique in that it supports the *entire* spectrum of SPM functionality by means of *software agents*. Thus the SPMSA model aims to address the shortcomings of current SPM practices as a result of the changing nature and dynamic changing SPM environment.

Table 7.2 below provides a summary of how the SPMSA model addresses the limitations of current SPM approaches through software agents.

Table 7.2 Limitations of SPM addressed by agent technology

Limitations of current SPM approaches	Agent enhancement
Environment	
Stakeholders in virtual teams may have different goals; different cultural backgrounds (Chen et al., 2003).	Virtual teams supported by automated agent interaction work toward a similar goal , with coordination and collaboration of team members.
Support communication in homogeneous environments and will need additional features or measures to connect heterogeneous elements (O'Connor and Jenkins, 1999).	Agents support heterogeneous environments, thus improving and enabling <ul style="list-style-type: none"> • communication and • coordination.
The system executes synchronously and must be connected to execute (Maes, 1994).	Agent system executes asynchronously and autonomously, thus <ul style="list-style-type: none"> • less network load, • less communication overhead.
Do not sufficiently support the knowledge representation of the SPM area (O'Connor and Gaffney, 1998).	Agent systems provide assistance with regard to knowledge management, namely knowledge of plans and designs, and can provide mechanisms to reason about these elements.

Limitations of current SPM approaches	Agent enhancement
Human interaction / automated control	
Documents are distributed by human action, thus there is the possibility of human error, such as omission (Purvis et al., 2003).	Workflow management to all relevant team members is automated , <ul style="list-style-type: none"> • documents and information are dispersed and • retrieved from the repository.
Team member interaction depends on user/human interaction, thus prone to errors (Benfield et al., 2006).	Automates team member interaction , by regular prompting for input to ensure that <ul style="list-style-type: none"> • the data is current and • tasks not forgotten, thus • improving productivity.
All actions, functions and coordination to be executed by team members, without specific process coordination measures (Petrie, Goldman and Raquet, 1999).	Automates process coordination , which will improve programmer productivity as well as minimise errors.
Tasks	
Complexity of tasks and environment one of the reasons for failure (Jennings, 2001; Benfield et al., 2006).	Complexity of tasks is minimised by automated support, such as calculations automated , thus <ul style="list-style-type: none"> • reducing complexity of the solution, • improving programmer productivity.
Large systems are difficult to maintain consistently over a set period of time. Current tools do not provide proper change notification (Petrie, et	Maintenance is automated and users are prompted for input on changes. Change control is automated and users are regularly prompted for input. Changes are incorporated dynamically.

Limitations of current SPM approaches	Agent enhancement
al., 1999).	
Current tools support the reporting and calculation facilities, but not continuous progress management (Chandrashekar et al., 1993).	<p>Management of progress status is automated, e.g. risk monitoring and risk status are checked on a daily basis, enabling the project manager to</p> <ul style="list-style-type: none"> • identify problems early and • take proactive measures (Roy, 2004).
According to a study by Verner and Cerpa (2005) risks are identified at the start of a project but only 50% follow the risk through during development.	Agents that monitor risk will automate the continuous monitoring of risks , thus following all risks throughout the project.
Different team members use passive corporate reporting aspects (Chen et al., 2003).	Continuous input of task status and sharing of information changes passive reporting to a system that supports dynamic reporting , thus improving coordination and cooperation between team members.
Data, tasks and results will be sent over a network to execute at the user's workstation (Chen et al., 2003).	Tasks are embedded in agent behaviour, thus by traversing the network, agents lessen communication overhead and network load because tasks execute at team member's site.
Ineffective and inefficient communication, i.e. untimely information, failure to notify all team members, not enough top-down information flow and storing information in a format not suitable for retrieval (Chen	Collaborative tool that provides automated support on structures for efficient information sharing, set format for information storing and structures for communication to promote adequate and timely information sharing (Gawinecki et al., 2007).

Limitations of current SPM approaches	Agent enhancement
et al., 2003).	
Quality measures and standards are selected by team. Human inaccuracy and omissions are possible (O'Connor and Jenkins, 1999).	Continuous automated input are received from all team members for quality control . Agents may supply measures and directives that conform to standards .
Current SPM tools provide no intelligent support on standards or best practices (O'Connor and Moynihan, 2000).	Agents with intelligence may encapsulate areas of experience such as standards, and may advise the project leader on best practices and standardisation. This provides knowledge base support (Gawinecki et al., 2007).
Intelligent support	
Bidding and negotiation are done by humans (Kephart and Chess, 2003; Badica, et al., 2008).	Advantages of bidding and negotiating agents . Automation of these functions will mean less work for the developer and added productivity for the developer/s.
Current SPM tools that make projections concerning tasks and decisions are static and do not support dynamic simulation (Joslin and Poole, 2005).	Agent systems support dynamic simulation concerning planning of uncertainty, i.e. dynamic resource allocation. Simulations may help the manager to anticipate critical conditions earlier and enable him to implement preventative measures – thus proactive SPM.
All interaction occurs through stakeholders but no support in decision-making process of project manager (O'Connor and Gaffney, 1998; Purvis et al., 2003).	Personal assistant agent supports each individual team member to intelligently manage and analyse large amounts of project data (Myers et al., 2007; Gawinecki et al., 2007).

7.4 CONCLUSION

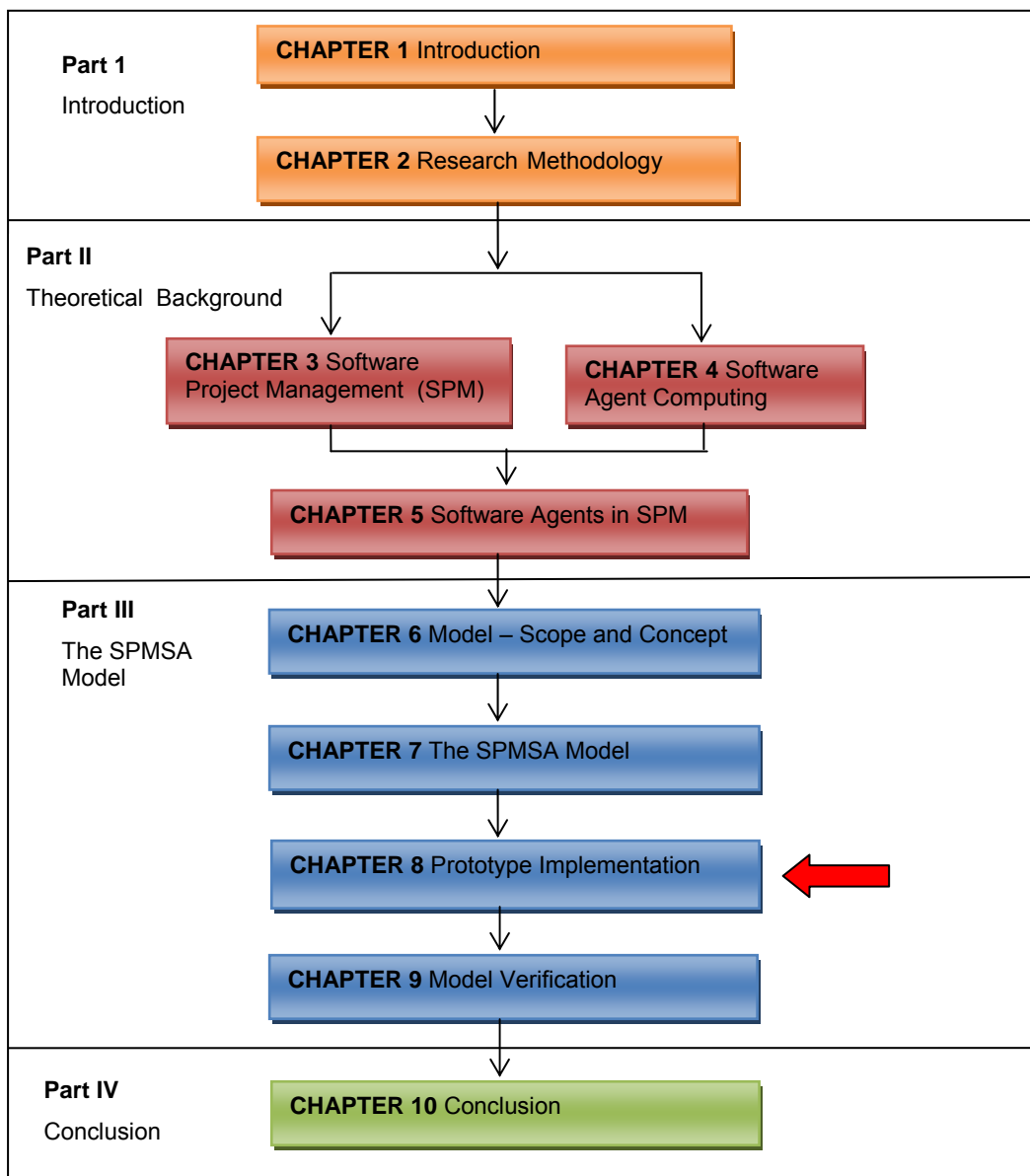
In this chapter the complete SPMSA model was compiled. This was accomplished by compiling a graphical representation of each of the SPM key functions that depict the different phases of each function. The researcher highlighted how a team of agents could support each phase.

The agent 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 SPMSA model enhances SPM by addressing the *entire* spectrum of SPM development by means of software agents. A prototype of a section of the SPMSA model is implemented as 'proof of concept' and will be discussed in the following chapter. The SPMSA model will also be verified against the PDCA cycle, as well as against the ISO 10006:2003 standard in order to substantiate its relevance.

CHAPTER 8

8 PROTOTYPE IMPLEMENTATION



8.1 INTRODUCTION

In previous chapters it was established that software agent technology is suitable to address the changing and ever-evolving SPM environment. A model, namely the SPMSA model, was consequently developed to support all SPM processes with a software agent framework. This chapter is devoted to a discussion of the development and implementation of a prototype of a section of the SPMSA model, namely the risk management function.

The purpose of this chapter is to show that the SPMSA model is not merely a theoretical concept, but that it can be implemented successfully. The prototype is used as 'proof of concept' to illustrate the possibility of using software agent technology to support SPM processes. The prototype could, of course, be expanded to implement the entire SPMSA model.

8.2 DEVELOPING THE PROTOTYPE

The SPMSA model supports the teams and individual team members in the SPM environment with a framework of software agents while executing their tasks. The project manager (also acting as team leader in this case), together with individual team members, will be supported during software project management processes to simplify their tasks, eliminate the complexities, and enhance coordination and communication. The prototype will specifically support the risk assessment part (i.e. the risk identification and the risk analysis and prioritisation phases) and the risk-monitoring phase of the risk control part of the SPMSA model.

The SPMSA model may be implemented by using agent black boxes in support of SPM functions. Each of the key processes discussed in Chapter 7 could successfully be addressed by following a black box approach that is based on software agent technology. Each black box consists of collaborative software agents that ensure 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 which each has a long list of tasks to accomplish. This implies that the complexity of creating and maintaining tasks could be greatly reduced.

The prototype specifically targets the *risk management function* of the SPMSA model, as was indicated earlier. The prototype is developed according to the basic phases of agent development as indicated in Chapter 5 (Sections 5.2.1.1 and 5.2.1.2), namely the requirements analysis phase and the design phase.

8.2.1 Requirements analysis phase

The prototype is used as 'proof of concept', therefore only some functions will be used to illustrate the concept of agent support. Implementing the full spectrum of the SPMSA model falls outside the scope of this thesis. However, the prototype is used as evidence that the SPMSA model can be implemented and can support and enhance the SPM arena.

The aim of the agent framework in the prototype is to support the project manager and team members in their tasks during *software risk management*. As mentioned in Chapter 5, different approaches to design agent systems have emerged, but there is currently not a single unified and accepted agent-orientated methodology (Iglesias et al., 1998; Zambonelli et al., 2001). In this study, object-oriented diagrams are adapted and used for the requirements analysis phase, with regard to the following:

- The requirements description
- A model, i.e. use cases to describe external interaction with the system and individual agent tasks
- A social agent model for global interaction.

8.2.1.1 The requirements description

The purpose of the requirements description is to provide the developer with a clear detailed understanding of the requirements of the system (Whitten, Bentley and Dittman, 2004). The prototype implements the two phases of *risk assessment*, i.e.

risk identification and risk analysis and prioritisation. Additionally, the prototype also implements the risk-monitoring phase of *risk control*. Although it was possible, the risk management planning and risk resolution phases of risk control were not implemented since the scope was considered too large for the purpose of this study. Both phases would also require extensive participation from the project manager concerning risk planning and resolution.

The purpose of each of the phases that are indeed implemented by the prototype can be summarised as follows:

- During the *risk identification phase* potential threats, discrepancies and overall inconsistencies concerning the project plan schedule, budget or time frame are identified. Risk identification will be done by both the team leader and team members.
- The purpose of the *risk analysis and prioritisation phase* is to assess the likelihood of a particular risk occurring, as well as a prediction on its potential impact in monetary value.
- Additionally, a list is produced of risk items ranked as top risks for this project.
- The *risk-monitoring phase* involves tracking the process of the project towards resolving its risk items in order to enable the team or project manager to take preventative measures or corrective action where appropriate. For instance the prototype monitors the task status of each team member.

8.2.1.2 The model

The purpose of the model is to indicate the interaction of external entities, such as the project manager and individual team members with the agents. The model is illustrated by means of a use-case diagram, in Figure 8.1 below.

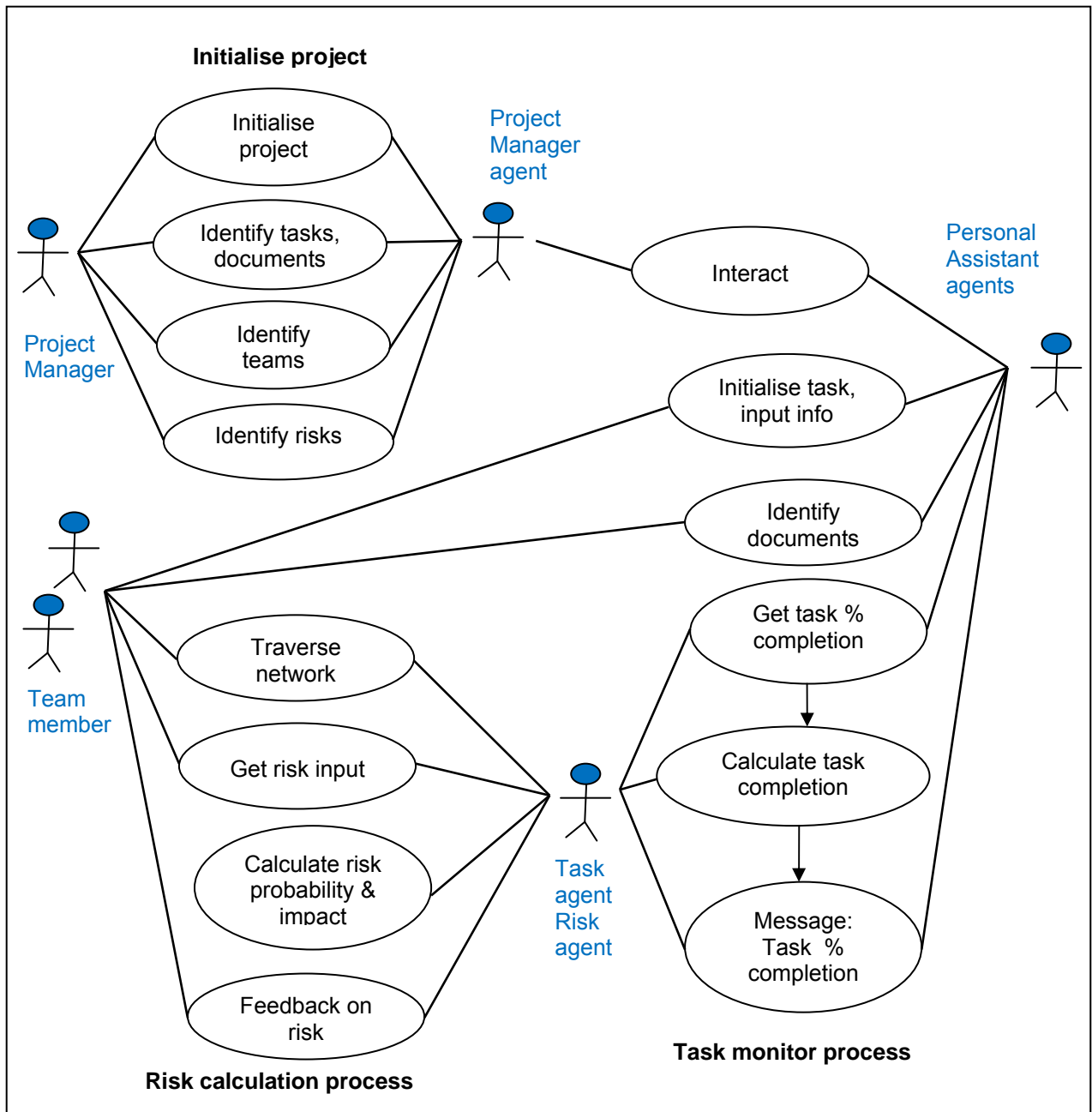


Figure 8.1 Use-case for the Jade Project Management Prototype System (JPMPS)

8.2.1.3 A social agent model

The purpose of the social agent model is to indicate the interaction of agents with the environment and with other agents. The conceptual view of the SPMSA model presented in Chapter 6 (Figure 6.2) illustrates both the phases of software development to be supported, and the software agent framework to support these

phases. The bottom section of Figure 6.2 comprises an agent framework that describes the different agents in the system, as well as their interaction with each other and with the team. Figure 6.2 was adapted to create the social agent model for the risk management function for this prototype (see Figure 8.2). This social agent model contains a risk agent as an additional agent to be utilised for this specific implementation, whereas the client agent and messaging agent are not utilised.

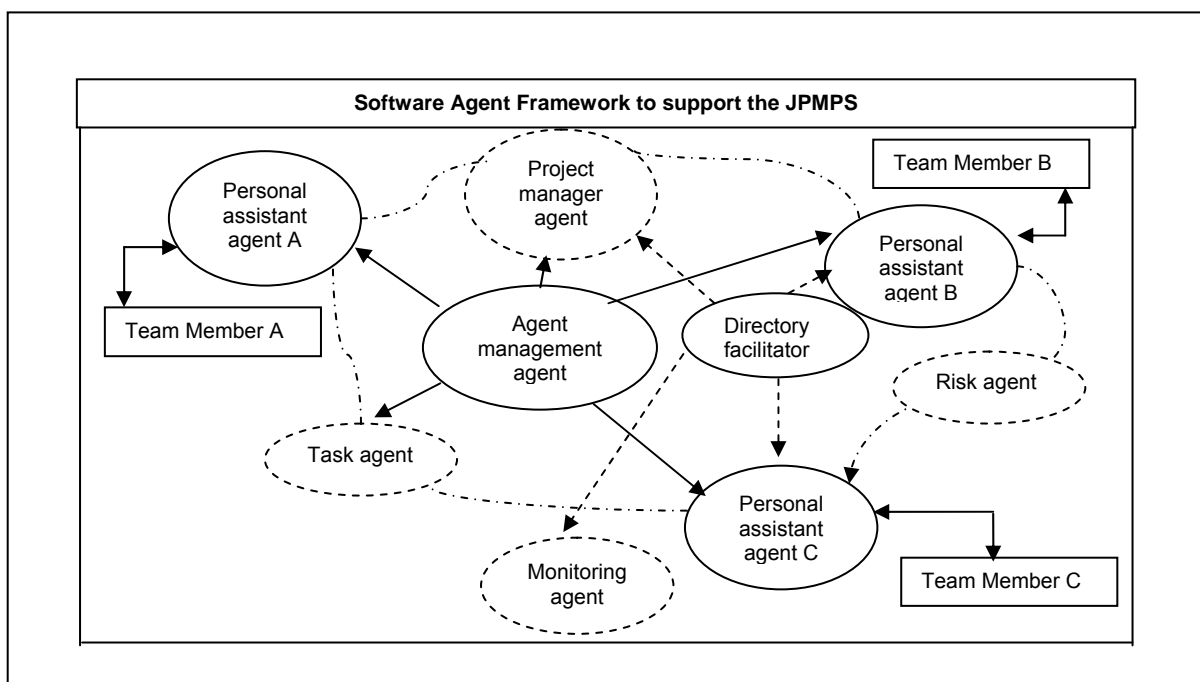


Figure 8.2 Social agent model for the JPMPs

The different agents used in this framework are the agent management agent, the directory facilitator, monitoring agent, personal assistant agents, the project manager agent, task agent(s) and risk agent(s) – as depicted in Figure 8.2

The interaction of the agents with each other is summarised in Table 8.1.

Table 8.1 Agent interaction with other agents

		AM	DF	MA	PA	PMA	TA	RA
JADE System	Agent management agent (AM)	X	X	X	X	X	X	X
	Directory facilitator (DF)	X	X	X	X	X	X	X
Agents	Monitoring agent (MA)	X	X	X		X	X	X
	Personal assistant agent (PA)	X	X			X	X	X
	Project Manager Agent (PMA)	X	X	X	X	X	X	X
	Task agent (TA)	X	X	X	X	X	X	X
	Risk agent (RA)	X	X	X	X	X	X	X

A description of the purpose of these agents are contained in Table 7.1 (Chapter 7).

8.2.2 Design Phase

The design should indicate the solution to the problem based on the outcome of the requirements analysis phase. The main focus of this phase is on *how* to resolve the problem. This is traditionally identified by an architectural design as well as a detailed design. In this case the social agent model indicates the architectural structure, as well as the relationship between agents to indicate inter-agent dependencies. The prototype is implemented in JADE, and JADE has an inbuilt mechanism for describing detailed agent interaction, thus this will be illustrated as part of the implementation.

To facilitate the rapid development of an agent-based system, JADE contains a number of predefined classes that can easily be extended for a specific application. One such object class is that of the Agent class that is extended by all implemented agents. Other useful classes include behaviour classes that allow the encapsulation of agent actions. Various other classes and extensions exist that assist with

debugging, communication, agent self-management, the development of graphical user interfaces and web services, to name a few (Bellifemine, Caire, Poggi and Rimassa, 2003; FIPA, 2003).

8.3 PROTOTYPE IMPLEMENTATION

8.3.1 The Technological Platform

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, it is viable to implement the risk management function area of the proposed model in Java (Wooldridge, 2002).

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. One feature of the JADE framework is the ability to abstract away from the details of agent communication. The JADE platform allows for easier communication by adhering to the FIPA standard for Agent Communication called FIPA-ACL (FIPA, 2003). 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 development while ensuring compliance to standards through a comprehensive set of system services and agents.

The efficiency of the JADE platform for agent development has been tested in a scenario where the number of agents and messages are increased, to test the efficiency of agent creation and scalability (Chmiel, et al., 2004b). These researchers found that JADE is a very efficient environment limited only by the limitations of the standard Java programming language. The environment does not introduce substantial overhead, and JADE scales well when messages and agent movement are substantially increased.

The Jade agent platform used for this implementation provides a fully integrated agent platform with an agent management system and a directory facilitator.

8.3.1.1 Agent Management System

The entire process of agent management (including agent mobility, suspension, awaking, creation and destruction) is handled by the Agent Management System (AMS) referred to in Jade as the remote agent management system, which is also a FIPA recommendation. The JADE agents exist in agent containers provided by the JADE agent platform, upon which the agents are given access to the functionality of the Jade agent platform.

As shown in Figure 8.3, the JADE container provides a context for agent existence and can be extended by other containers to form a distributed environment.

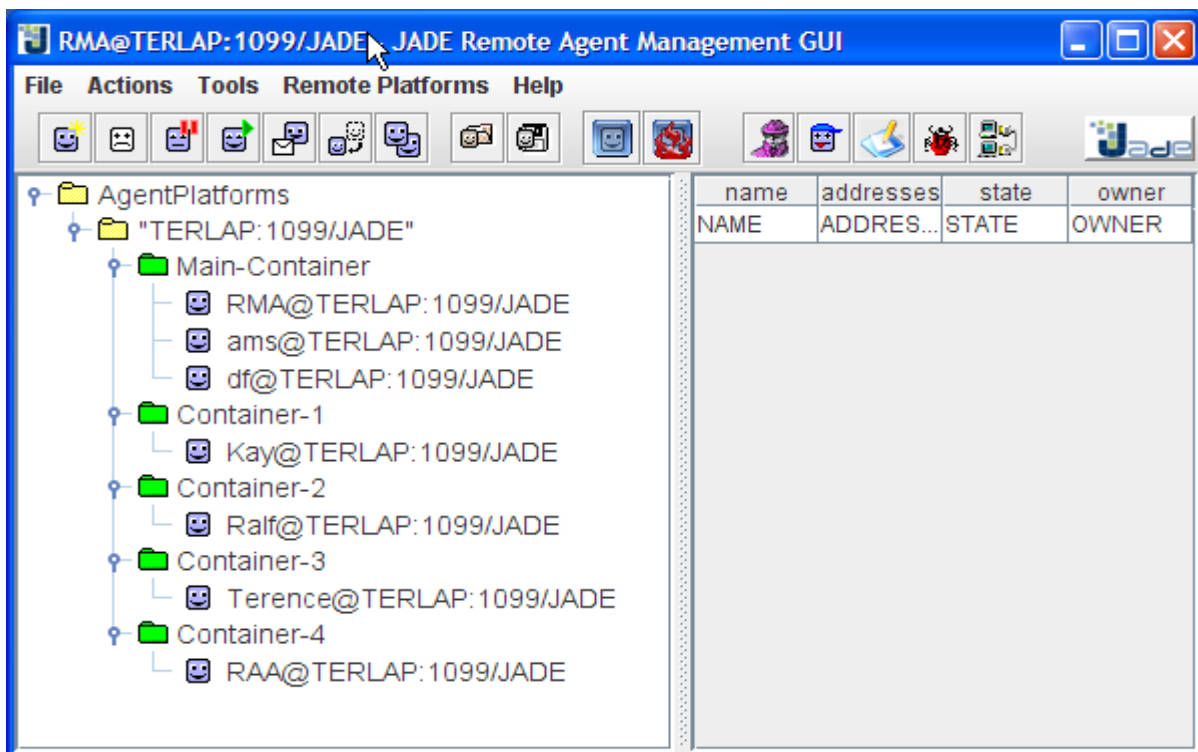


Figure 8.3 Agent management system

The agent management system – also referred to as agent management agent – is responsible for managing the team of agents, ensuring coordination between the

sub-tasks, communication between agents and the location distribution of agents. The agent management system provides the unique agent identifiers used to identify agents within the FIPA standard. In Figure 8.3 the Main-Container contains agents RMA@TERLAP: 1099/JADE, ams@TERLAP: 1099/JADE and df@TERLAP: 1099/JADE, while containers 1, 2, 3 and 4 each contains one agent.

The agent management system also provides communication interfaces in which agents can exist. Agents register with this agent management system and it provides execution cycles and mobility for the agents. Upon registration each agent receives a globally unique identifier. The agent management system provides a white pages service, which provides a listing of available agents with each one's address. The agent management system is available to the system without additional programming, as it is already a feature of the JADE platform.

8.3.1.2 Directory Facilitator

A second feature of agent interaction that is explicitly handled by the JADE framework and that also adheres to the FIPA recommendations for an agent-based platform, is that of the Directory Facilitator (DF). The DF acts as a yellow page service to enable the discovery of agents and the relevant services provided by said agents, as illustrated in Figure 8.4.

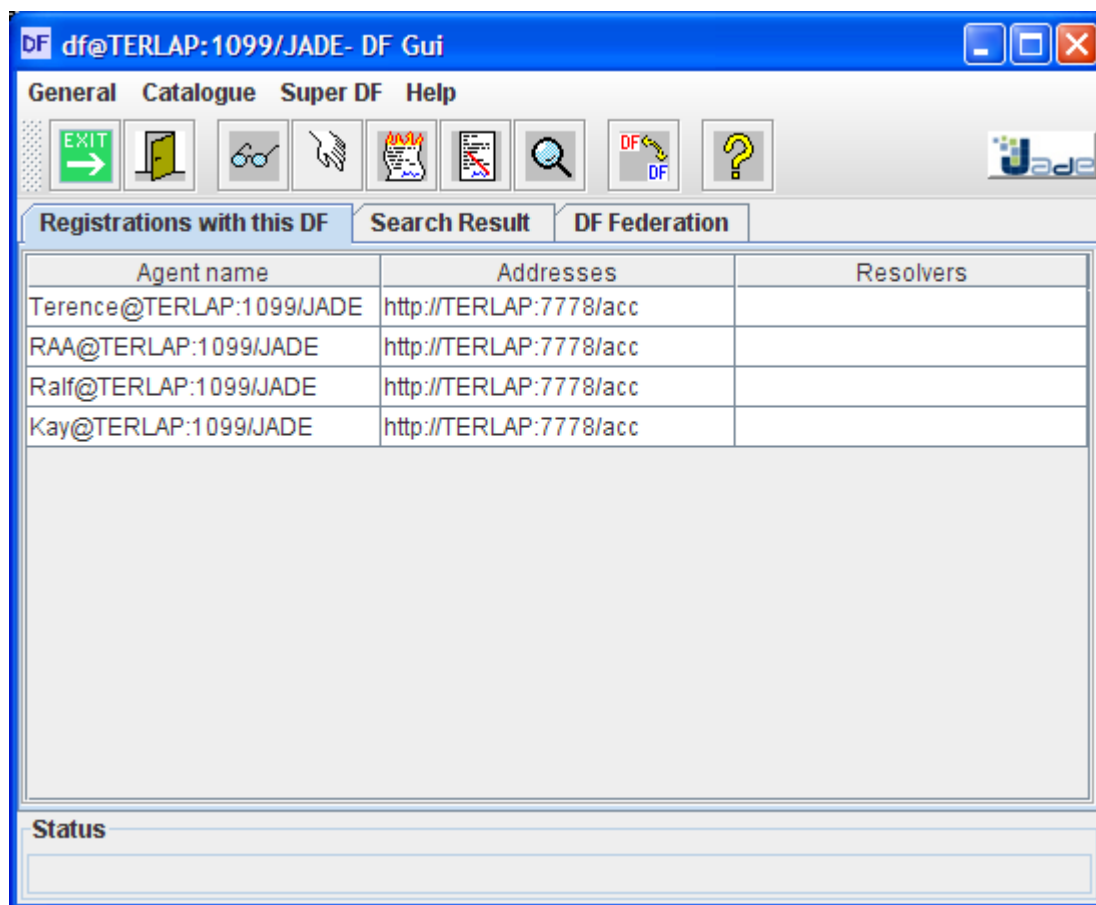


Figure 8.4 Directory facilitator

The DF in Figure 8.4 has four agents registered. The first agent is identified by a name – Terence@TERLAP:1099/JADE, and address – http://TERLAP:7778/acc. The remaining agents are RAA@TERLAP:1099/JADE, Ralf@TERLAP:1099/JADE and Kay@TERLAP:1099/JADE, each with its own address.

The directory facilitator is also available to the system without additional programming, because it is a feature of the Jade agent platform.

8.3.2 Overview of the prototype

The prototype supports the software project manager in his/her task during the risk identification, risk analysis and prioritisation phases, as well as during the risk-monitoring phase. The agents collaborate in a distributed environment to achieve the overall objective of software project management (Nienaber, Smith, Barnard and

Van Zyl, 2008). Various roles are captured in reusable entities called “behaviours”, which can be assigned to agents at design time.

The prototype entitled “The Jade Project Management Prototype System” (JPMPS) was developed at the Meraka Institute (African Advanced Institute for Information and Communication Technology). The researcher compiled the specifications for the prototype, including the use-case diagram and the social agent model. The coding for the prototype was done at Meraka. The researcher tested the prototype and suggested changes and enhancements. Consequently the researcher compiled Chapter 8, with input from the developer at Meraka, pertaining to the technological platform (section 8.3.1) as well as the identification of the advantages of the prototype after it was used for a true-life system (section 8.3.3).

The prototype was accordingly tested with the Corridor Sensor Web Application by the team leader of the project. The Corridor Sensor Web Application (CSWA) is a small in-house project that aims to deploy a test bet for Sensor Web type applications. The project team consisted of three members. The screenshots that follow all relate to the prototype implemented in the CSWA project environment.¹

8.3.2.1 Risk Identification

The aim of the *risk identification phase* is to identify, through interaction with all team members, specific risks that may pose a problem. Feedback concerning identified risks will be *shown* to the project team on a *continuous basis*.

The application is activated by double clicking on the **ProJectMan** icon on the desktop. This provides the project manager with an input screen to start the project, as shown in Figure 8.5 – the initial screen will contain no data in the **Project Name**, **Tasks**, **Risks** and **Team Members** edit boxes.

¹Permission was obtained to publish the results of testing the prototype in the said project.

The screenshot shows a software window titled "Team Lead Project Manager". It has a standard Windows-style title bar with minimize, maximize, and close buttons. The main content area is divided into several sections:

- Project Name:** A text input field containing "Corridor Sensor Web".
- Tasks:** A list box containing five items: "Develop Data Adaptor", "Develop Sensor Observation Service", "Develop Sensor Planning Service", "Deploy Sensor Web Application", and "Test Sensor Web Application". To the right of this list is an "Add Task" button.
- Risks:** A list box containing five items: "Personnel Shortfalls", "Inadequate Knowledge/Skills", "Unrealistic time estimates", "Unrealistic cost estimates", and "Developing wrong software functions".
- Team Members:** A list box containing three items: "Denise", "Wabo", and "Terence".

At the bottom right of the window is a "Save" button.

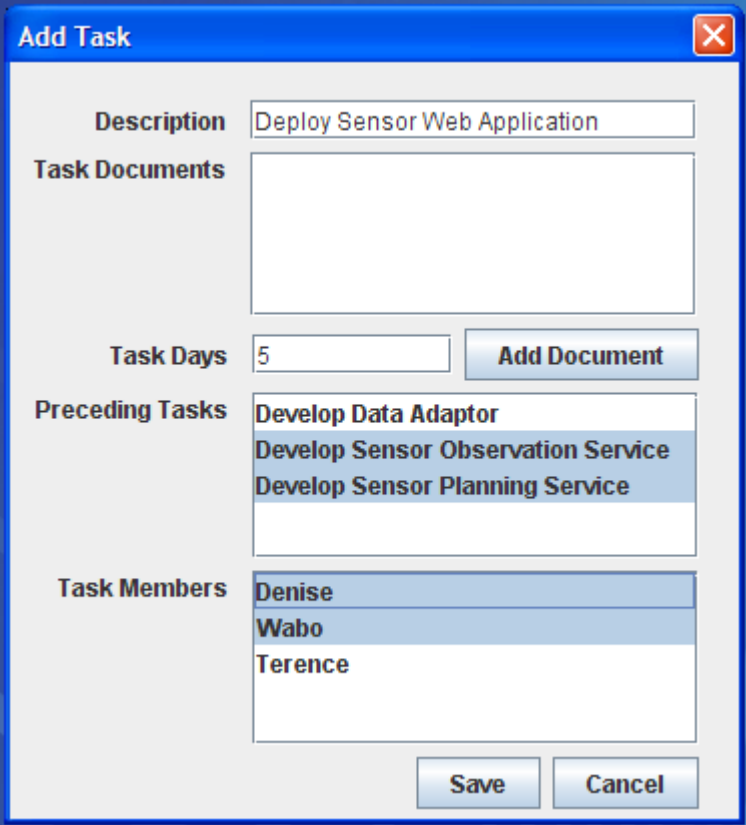
Figure 8.5 Input screen for the CSWA project

The *first step* will be to *initialise the specific project* by entering an identifying project name (in this case **Corridor Sensor Web**) and team members (i.e. **Denise**, **Wabo** and **Terence**) into the **Project Name** and the **Team Members** edit boxes respectively – see Figure 8.5. For each of these team members a **personal assistant agent** is immediately activated by the system.

The *second step* will be to *select the risks* for this project. The project manager selects the relevant risks from the list of risks displayed in the **Risks** list box by clicking on the appropriate risk – see Figure 8.5.

The *third step* will be to *identify the tasks* for this project. The identified tasks will be entered by the project manager by using the **Add Task** button on the input screen displayed in Figure 8.5. The project may have several tasks, while each one will have a timeframe (in days) for completion and possibly one or more preceding tasks.

When clicking the **Add Task** button in Figure 8.5, the screen depicted in Figure 8.6 will be displayed – the initial screen will contain no data in the **Description**, **Task Documents**, **Task Days**, **Preceding Tasks** and **Task Members** edit boxes.



Description	Deploy Sensor Web Application
Task Documents	
Task Days	5 <input type="button" value="Add Document"/>
Preceding Tasks	Develop Data Adaptor Develop Sensor Observation Service Develop Sensor Planning Service
Task Members	Denise Wabo Terence

Figure 8.6 Task: Deploy Sensor Web Application

An example of adding the information for one of the tasks, i.e. the **Deploy Sensor Web Application** task, is displayed in Figure 8.6. The name of the task, namely **Deploy Sensor Web Application**, is entered in the **Description** edit box. The task documents, i.e. a list of specifications concerning the task, can be added with the **Add Document** button, and will then appear in the **Task Documents** edit box. This task currently has no task documents added. It has been allocated five working days (indicated next to the **Task Days** edit box) to be completed. The other tasks namely **Develop Data Adaptor**, **Develop Sensor**

Observation Service and **Develop Sensor Planning Service** are displayed in the **Preceding Tasks** list box. Finally, the list of team members available for this project is displayed in the **Task Members** list box. **Denise** and **Wabo** will be responsible for this specific task, as indicated by the shading in Figure 8.6. Information will be saved or cancelled by the **save** and **Cancel** buttons respectively at the bottom of this input screen. Information for the other tasks related to the current project will be entered in a similar way. All tasks are keyed in at the beginning of the project as shown in Figure 8.5.

Each member of the project team will be supported by a **personal assistant agent**, which will reside on each team member's computer desktop. Such an agent will provide information concerning the task allocated to the team member. Figure 8.7 illustrates **Wabo's personal assistant agent**, concerning task **Develop Sensor Planning Service**, which is on schedule.

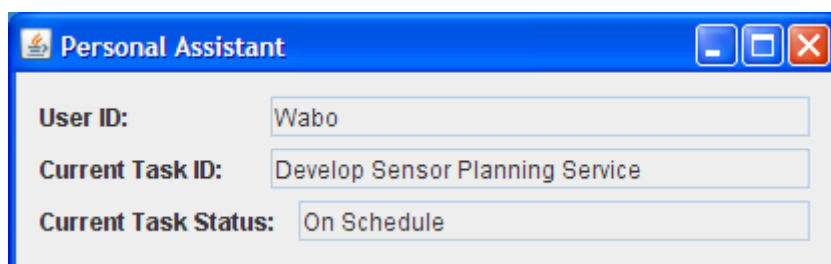


Figure 8.7 Personal Assistant Agent: Wabo

Such an agent screen will appear *automatically* on the desktop computer of each team member.

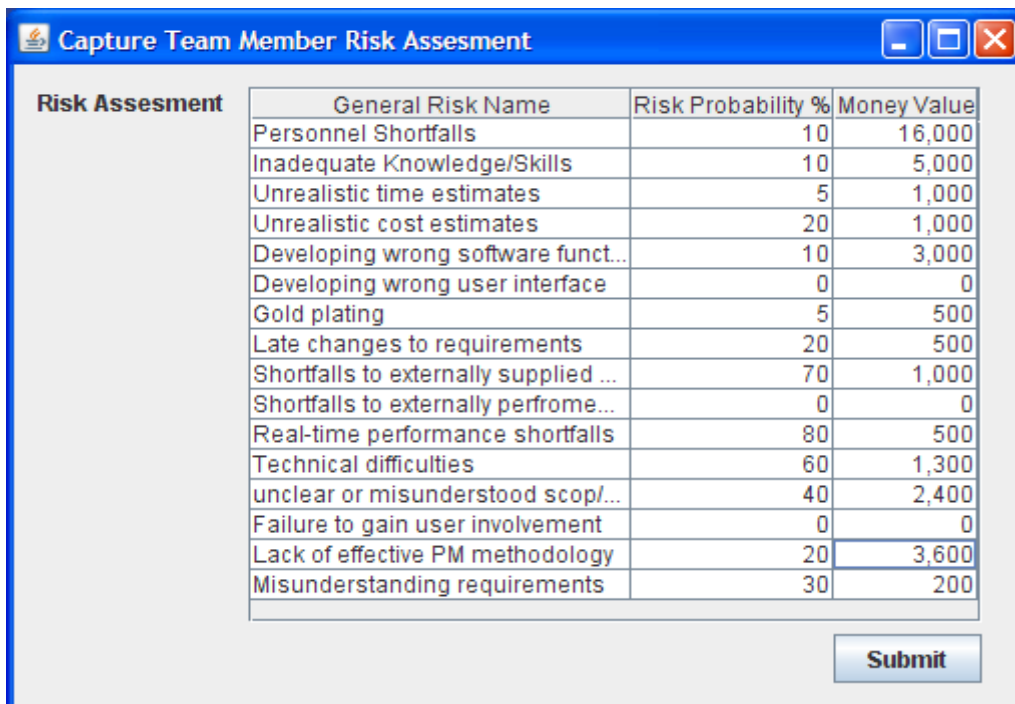
The **personal assistant agent** of each team member will support him/her in daily planning tasks such as setting up meetings, compiling and distributing an agenda, attaching required documents and finalising the place and time. The personal assistant agent will have information about tasks of the team members it is associated with and will be able to support the team member by monitoring the time set to complete the task. Documents will be appended and the user will be

prompted when documents should be attached. The workflow is thus automated, input for tasks is automated, the progress is monitored and relevant information is sent to the **monitoring agent**. Change control will also be monitored and addressed. The team may be geographically dispersed, but the heterogeneous system will address the discrepancies in technology and area.

8.3.2.2 Risk Analysis and prioritisation

During the *risk analysis and prioritization phase*, the **task agent** will *traverse the network* of team members, visiting each **personal assistant agent** of each team member.

A risk input screen, similar to the screen depicted in Figure 8.8, will *automatically* pop up on the desktop computer of each team member for each task related to the specific team member, *at regular intervals*, i.e. the information depicted on this screen will continuously be updated.



The screenshot shows a window titled "Capture Team Member Risk Assessment" with a table of risk data. The table has three columns: "General Risk Name", "Risk Probability %", and "Money Value". The data is as follows:

Risk Assessment	General Risk Name	Risk Probability %	Money Value
	Personnel Shortfalls	10	16,000
	Inadequate Knowledge/Skills	10	5,000
	Unrealistic time estimates	5	1,000
	Unrealistic cost estimates	20	1,000
	Developing wrong software funct...	10	3,000
	Developing wrong user interface	0	0
	Gold plating	5	500
	Late changes to requirements	20	500
	Shortfalls to externally supplied ...	70	1,000
	Shortfalls to externally perfrome...	0	0
	Real-time performance shortfalls	80	500
	Technical difficulties	60	1,300
	unclear or misunderstood scop/...	40	2,400
	Failure to gain user involvement	0	0
	Lack of effective PM methodology	20	3,600
	Misunderstanding requirements	30	200

A "Submit" button is located at the bottom right of the window.

Figure 8.8 Risk probability and monetary value input: example 1

The screen depicted in Figure 8.8 displays all the risks identified for the specific task at hand. Each team member responsible for this task will receive such a screen – with no data initially. The team member will allocate a percentage value to indicate the probability of a specific risk occurring. In addition, the monetary value of the risk – if it were to occur – should also be entered if possible. For example, in Figure 8.8 the probability of **Personnel shortfalls** to occur is 10%, and if this risk occurs, it will result in a cost of R16 000. Figure 8.9 illustrates another example of a team member's input.

General Risk Name	Risk Probability %	Money Value
Personnel Shortfalls	20	18,000
Inadequate Knowledge/Skills	15	6,000
Unrealistic time estimates	10	700
Unrealistic cost estimates	30	700
Developing wrong software funct...	15	2,600
Developing wrong user interface	0	0
Gold plating	7	450
Late changes to requirements	10	550
Shortfalls to externally supplied ...	60	1,200
Shortfalls to externally perfrome...	50	0
Real-time performance shortfalls	40	600
Technical difficulties	25	1,300
unclear or misunderstood scop/...	25	3,000
Failure to gain user involvement	0	0
Lack of effective PM methodology	15	4,000
Misunderstanding requirements	35	500

Figure 8.9 Risk probability and monetary value input: example 2

The **task agent** then *continuously* calculates and presents the average percentage of a specific risk occurring, given the input values of *all* team members. The risk probability is calculated as the (probability of occurrence i.e. risk probability percentage) x (potential damage i.e. potential damage in monetary value) (Hughes and Cotterell, 2006).

Furthermore, each team member is *prompted* automatically on a daily or regular basis (decided by the designers of the system) for input on the percentage of task completion (for their specific task in hand) at that time. The system compares this to the set date for completion. Consider for example the screen depicted in Figure 8.10 below.

General Risk Name	Risk Probability %	Money Value
Personnel Shortfalls	15	18,000
Inadequate Knowl...	12	5,000
Unrealistic time es...	7	733.333
Unrealistic cost est...	24	766.667
Developing wrong ...	10	2,633.333
Developing wrong ...	11	0
Gold plating	9	456.667
Late changes to re...	10	483.333
Shortfalls to extern...	45	1,166.667
Shortfalls to extern...	21	0
Real-time perform...	56	533.333
Technical difficulties	28	1,400
unclear or misund...	38	2,600
Failure to gain use...	0	0
Lack of effective P...	16	3,666.667
Misunderstanding ...	33	366.667

Figure 8.10 Task completion input screen

Each team member should *key in the percentage* of the task that is completed in the **Task % Complete** window, as indicated in Figure 8.10. For example, the **Develop Sensor Planning Service** task is 50% complete. Note that the **risk probability values** at this stage reflect the average values (based on the input of all team members) and are displayed on this screen, and cannot be changed by the team member here. These values can only be changed on the risk probability and monetary value input screen (depicted in Figures 8.8 and 8.9) as discussed earlier.

The **risk agent** will continuously monitor the status of all tasks, and this will keep the team members conscious of the time aspect. The **risk agent** thus traverses the project environment, as shown in Figures 8.11 and 8.12.

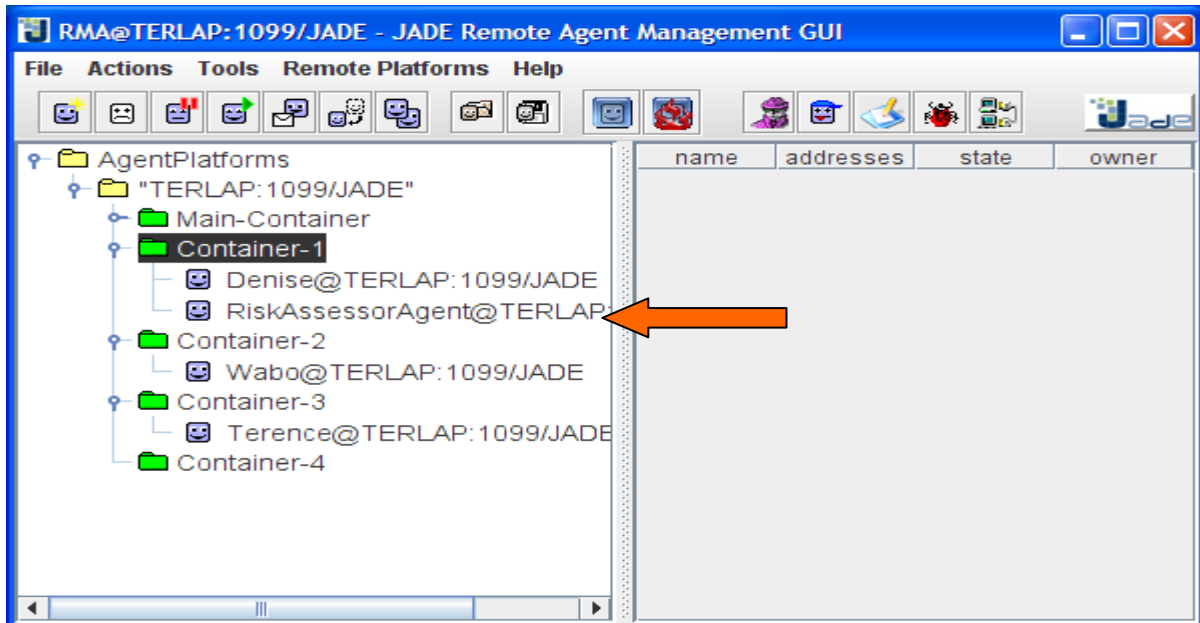


Figure 8.11 Risk agent in Container-1

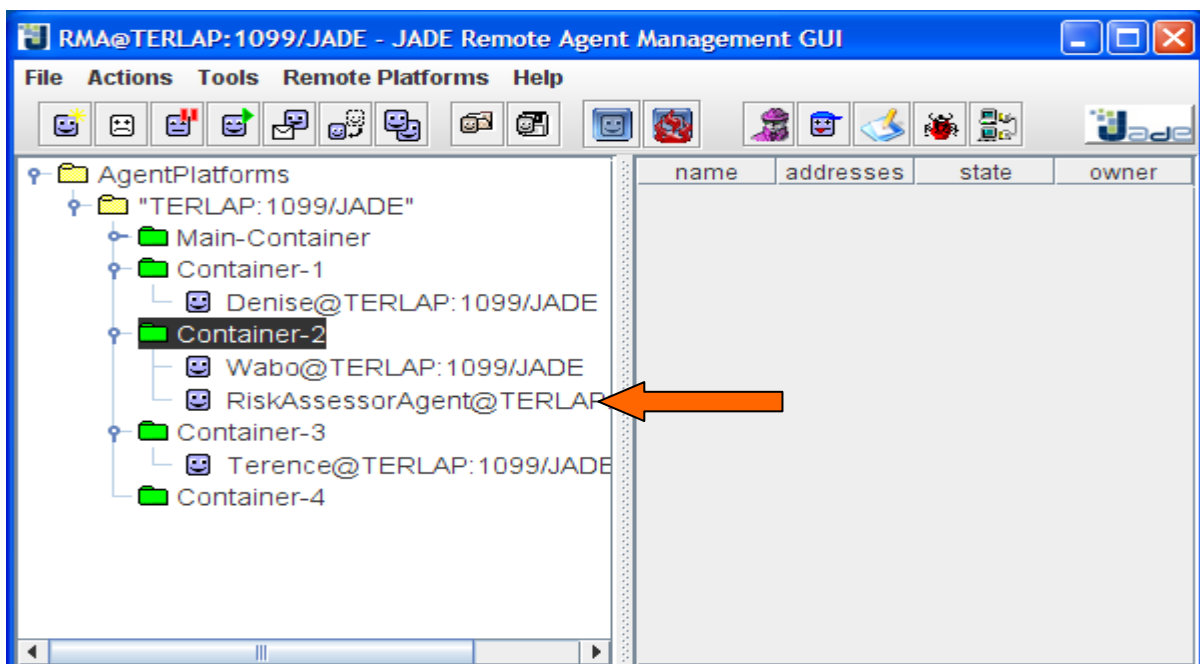


Figure 8.12 Risk agent in Container-2

Figure 8.11 shows the **risk agent** in Container-1, getting input from Denise (see arrow) regarding percentage task completion, whereas Figure 8.12 illustrates that the **risk agent** has moved to Container-2, where it is getting input from Wabo (see arrow) regarding the percentage of completion of his task. Thus, the **risk agent** moves from team member to team member at set time intervals to get information about, firstly, the risks this specific team member considers possible of occurring, and, secondly, the percentage of the task allocated to this team member that has been completed.

A team member might login at a remote site or at another laptop, but the agent system will locate the team member. The **personal assistant agent** of each team member will continue to support the specific team member, regardless of the location of the specific computer, and it will continuously monitor the task completion. For example, Figure 8.13 indicates that the **Current Task Status** of the **Test Sensor Web Application** task that **Terence's personal assistant agent** is responsible for, is **Overdue**.

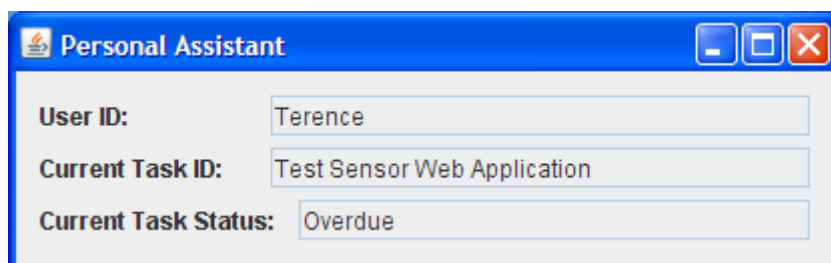


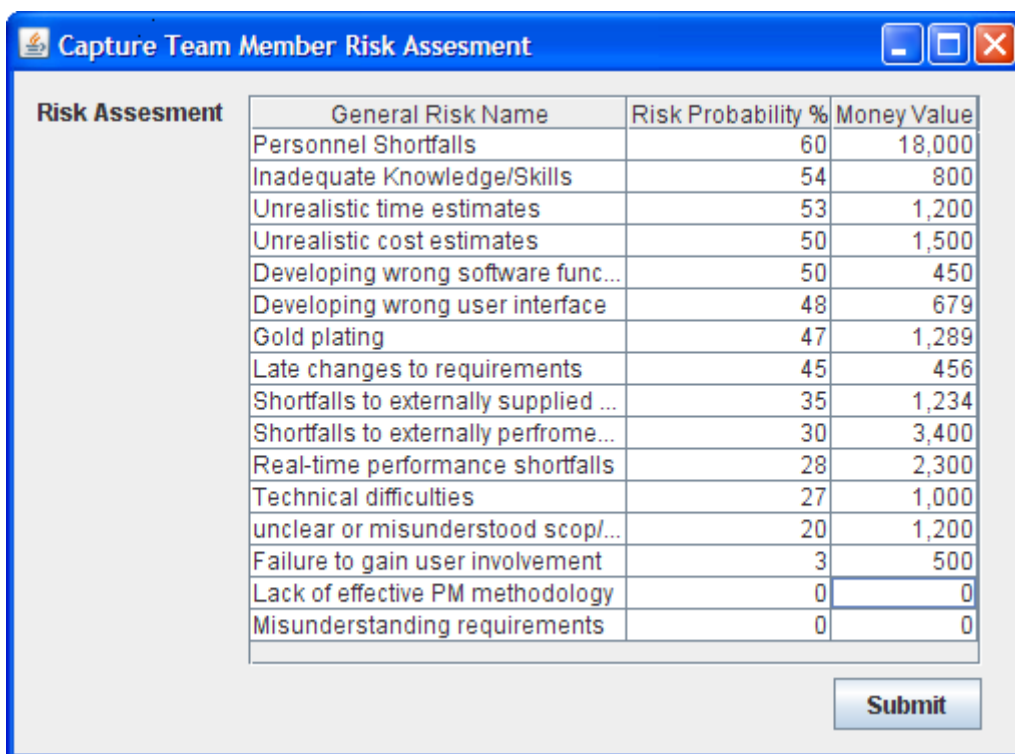
Figure 8.13 Personal Assistant Agent: Terence

Furthermore, the **task agent** will traverse the network and perform tasks needed, such as continuously getting input concerning risks identified from each team member, calculating the risk probability and distributing documents. Calculations on risk are automated by the **task agent** and any changes, such as one task taking longer than planned, will automatically be incorporated in the planning.

The **agent management agent** will continuously monitor all agent locations. Collaboration is thus attained through an agent-supported framework, the progress

being automatically monitored by the **monitoring agent**, and change control being executed when necessary. The **directory facilitator** will sustain a library for services for the agents to access if necessary.

During **risk prioritisation** the **task agent** will monitor the risks, and while traversing the environment, it will show the risks as prioritised to the team members. In other words, the team will continuously be provided with information on risk prioritisation. Figure 8.14 illustrates an example of risk prioritisation.



Risk Assessment	General Risk Name	Risk Probability %	Money Value
	Personnel Shortfalls	60	18,000
	Inadequate Knowledge/Skills	54	800
	Unrealistic time estimates	53	1,200
	Unrealistic cost estimates	50	1,500
	Developing wrong software func...	50	450
	Developing wrong user interface	48	679
	Gold plating	47	1,289
	Late changes to requirements	45	456
	Shortfalls to externally supplied ...	35	1,234
	Shortfalls to externally performe...	30	3,400
	Real-time performance shortfalls	28	2,300
	Technical difficulties	27	1,000
	unclear or misunderstood scop/...	20	1,200
	Failure to gain user involvement	3	500
	Lack of effective PM methodology	0	0
	Misunderstanding requirements	0	0

Figure 8.14 Risk prioritisation report screen

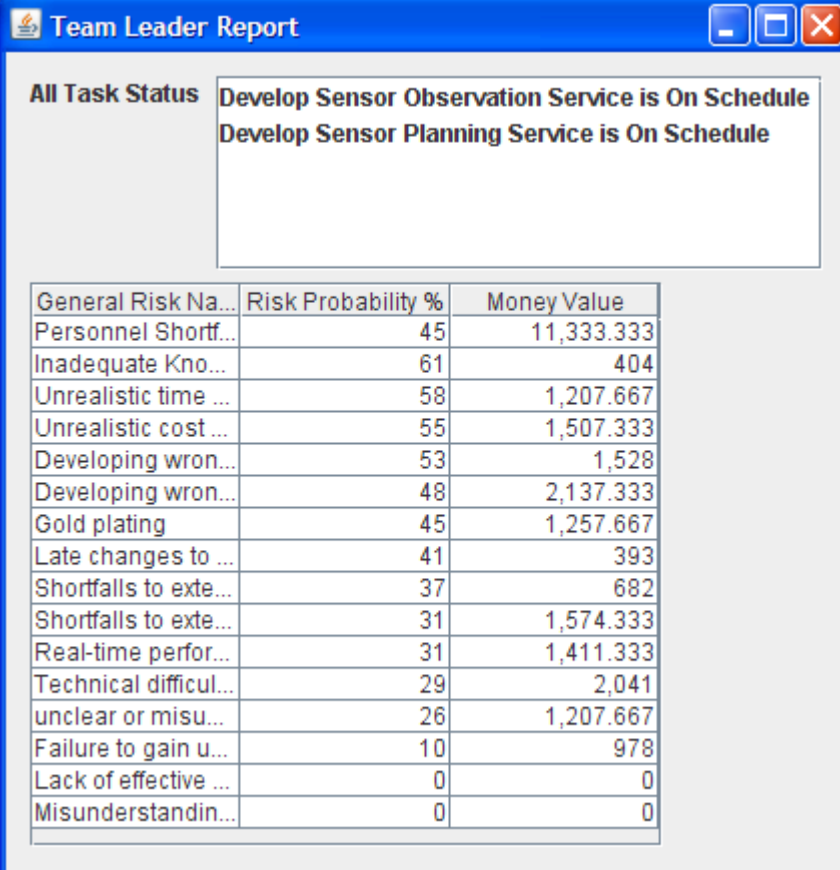
In Figure 8.14 the probability of **Personnel Shortfalls** occurring is the highest, i.e. 60% and therefore this risk is displayed at the top of the screen.

The risk screen will appear on each team member's desktop at *regular intervals*. At the same time the **personal assistant agent** of each team member will continuously display the task status on the screen, together with relevant messages

(as shown in Figures 8.7 and 8.13) to enable the team member to take action when necessary. Thus, interaction is forced and continuously sustained.

8.3.2.3 Risk monitoring

During the *risk-monitoring phase* the system will provide information on the status of all the team members' tasks, which will be shown on each member's desktop by his/her **personal assistant agent**. The **monitoring agent** will monitor tasks and report back to the **personal assistant agents**, where rescheduling of tasks as well as the notification of stakeholders can take place.



General Risk Na...	Risk Probability %	Money Value
Personnel Shortf...	45	11,333.333
Inadequate Kno...	61	404
Unrealistic time ...	58	1,207.667
Unrealistic cost ...	55	1,507.333
Developing wron...	53	1,528
Developing wron...	48	2,137.333
Gold plating	45	1,257.667
Late changes to ...	41	393
Shortfalls to exte...	37	682
Shortfalls to exte...	31	1,574.333
Real-time perfor...	31	1,411.333
Technical difficul...	29	2,041
unclear or misu...	26	1,207.667
Failure to gain u...	10	978
Lack of effective ...	0	0
Misunderstandin...	0	0

Figure 8.15 Team Leader Report

Tasks are monitored and an output screen (see Figure 8.15) will be shown, which illustrates the Team Leaders' screen, reporting that two tasks, **sensor Observation Service** and **Sensor Planning Service** are on Schedule.

The **risk probability** and **money value** are shown and cannot be edited at this stage.

Thus, the team leader will be supported by receiving information on a daily basis on risks prioritised, as well as a report on the percentage of each task that has been completed for all team members. Relevant messages as well as warnings will be given to attend to the problem. Tasks running late can be addressed, especially if the problem concerns a preceding task to another task. The necessary steps may then be taken. Action can also be taken to address the problem before the delay gets out of hand. Workflow is automated, documents are stored and updated automatically, task progress is monitored and change control is automated. The reporting facility can be extended to include PERT estimations and to calculate Z value and other project management calculations (Müller et al., 2004).

8.3.3 Outcome of using JPMPS

The Jade Project Management Prototype System (JPMPS) was used to manage the Corridor Sensor Web Application (CSWA) at the Meraka Institute. Having used the prototype, the team leader reported substantial benefits of using the prototype for the CSWA project. The JPMPS prototype supported and enhanced SPM processes, as opposed to other traditional SPM systems. Its benefits as experienced by the project leader are summarised below:

- All tasks have been successfully orchestrated, in other words the agent team successfully traversed the team environment, prompting each team member for input on task completion, as well as for documentation to accompany tasks. Thus, the cooperation of tasks between members was attained. Task members were relieved of the task to remember to send relevant deliverables or documents to other members. Agent teams automatically prompted team members to remind them of completion dates of tasks and deliverables, i.e. documents. Task completion of the entire project was monitored and reported on, thus supporting the project manager or team leader. This

enabled him/her to take corrective action before a task got out of hand or to spend more time on other functions.

- The whole team was alerted to the risks by means of the agent system continuously traversing the network and monitoring risks. Each team member, as well as the project manager, was kept up to date with the probability of risks occurring. In traditional applications risks can be analysed and visualised, but this has to be executed additionally by a team member. In this application the agent system autonomously executes this action and reports to the team.
- The team leader was able to keep track of any outstanding tasks, through the continuous feedback received from the agent system. Personal assistant agents provided feedback to team members with regard to task completion. Traditional applications will not provide this information on its own, without the team contacting or emailing each other.
- Overall, the use of an agent-based system with mobile agents added value to the project as no central server was required and the system was robust enough to cope with team members' working on different PCs and other network and infrastructure failure. The use of the agent system improves the adaptability of the project to changes in the environment.
- Team members were also not confined to a single working area or PC, but could log in at any other computer (at another site) and connect to the Internet, and the software agents would be able to track users to their new workstations.

8.4 CONCLUSION

In this chapter, the JPMPS prototype was discussed.

The prototype was tested in a real-life project i.e. the Corridor Sensor Web Application (CSWA), at the Meraka Institute. The project is of a very technical nature and relies on third party components regarding both hardware and software that had already been developed.

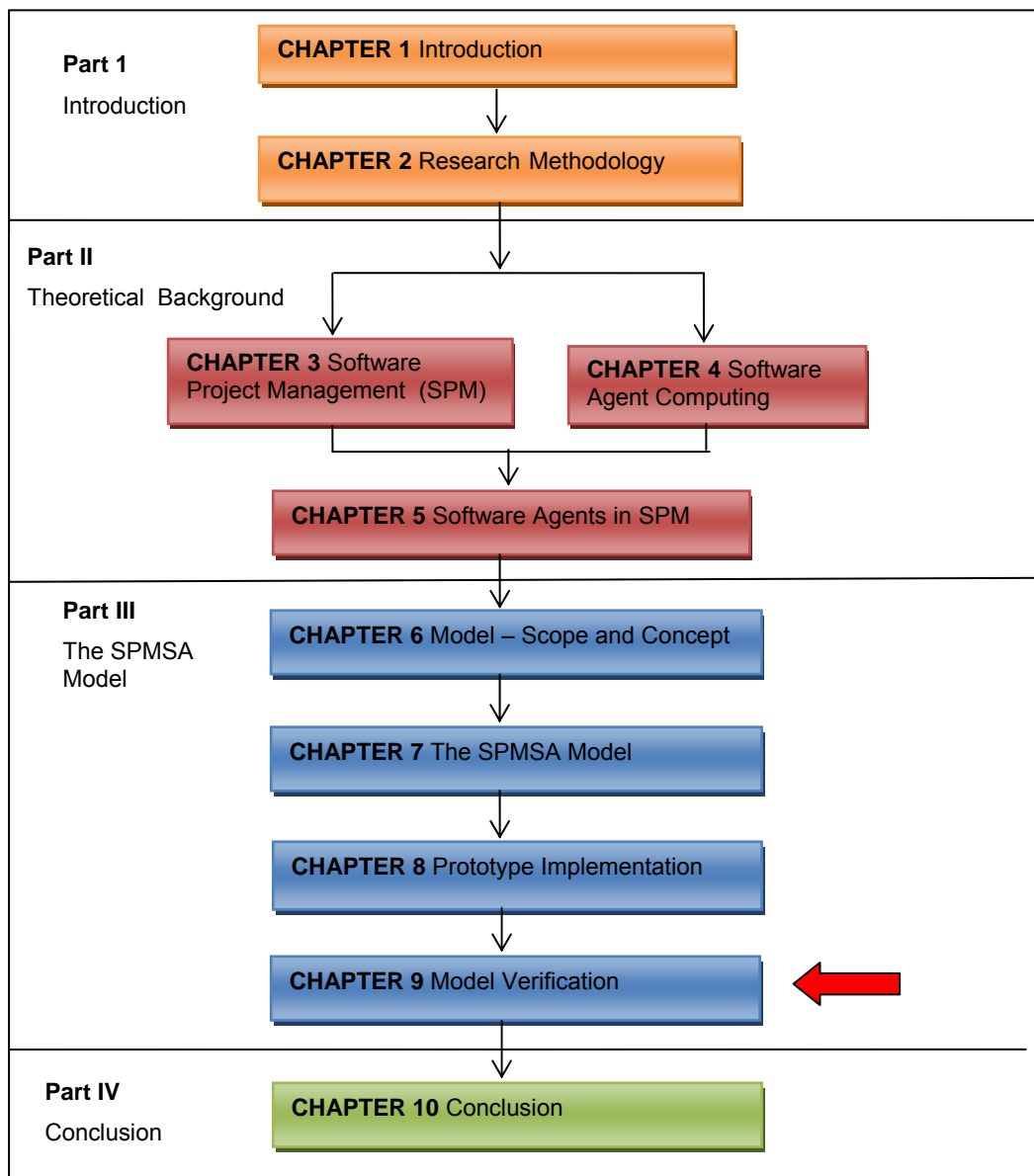
Using the prototype enabled the project leader to monitor risk probability and consequence, as well as the status of tasks and deliverables of the project. The SPM risk-monitoring processes were supported and enhanced by the team of agents that prompted team members for input on the status of tasks and deliverable. In this way the coordination and cooperation of tasks between members was attained. The autonomous functioning of the agent team also supported the team by provided feedback on each team members' task status and risk probabilities. It can therefore be concluded that the JPMPS succeeded in supporting and enhancing the SPM processes during risk management.

The JPMPS prototype currently implements only part of the risk management function as it is implemented as 'proof of concept', but it can be expanded to include all phases of risk management, as well as the entire SPMSA model, covering all areas of SPM.

In the next chapter, the SPMSA model will be substantiated by comparing the SPM phases of the model to the Plan-Do-Check-Act (PDCA) cycle, as well as to the ISO 10006:2003 standard.

CHAPTER 9

9 MODEL VERIFICATION



9.1 INTRODUCTION

The previous chapter was devoted to a discussion of the prototype that was implemented as ‘proof of concept’ to illustrate the possibility of using software agent technology to support SPM processes. The prototype supports the phases of risk assessment, namely risk identification and risk analysis and prioritisation, as well as risk monitoring, which is part of risk control. As stated before, the prototype can be expanded to include the entire risk management function and the entire SPMSA model, thus covering all areas of SPM. It is clear from previous chapters that agent technology will hold specific advantages for the SPM environment. This chapter is devoted to a discussion of the verification of the SPMSA model.

Table 3.2 (Chapter 3) contains the correlating phases of the core and facilitating functions of SPM, as compiled by the researcher. These phases, which form the basis of the SPMSA model, were compiled into a graphical representation in Figure 6.2 (Chapter 6) that depicts the *generic phases of software development* for each SPM key function.

The aim of this chapter is to evaluate and substantiate the SPMSA model. The SPM phases of the SPMSA model are therefore compared to the Plan-Do-Check-Act (PDCA) cycle. In addition, these phases are compared to the basic phases of software development as prescribed by the ISO 10006:2003 standard for projects.

9.2 ISO STANDARDS

Standards enable the adoption of standard project management practices, and can be defined as something established by authority, custom or general consent as a model or example (Garcia, 2005). The International Organization for Standardization (ISO) was formed in 1947 “to facilitate the international coordination and unification of industrial standards” (Garcia, 2005). In the 1990s, organisations responded to a fairly large number of industry standards (Schwalbe,2006). *Project management* standards, however, were contained as part of various standards, not all reflecting purely on software project management processes, but containing SPM

as part of a larger standard that deals with an engineering or information technology domain (Garcia, 2005).

Thus, over time several products appeared in an effort to contribute towards the adoption of standard project management practices, for example the Capability Maturity Model, Guide to Project Management Body of Knowledge (PMBOK), and the various ISO and ISO/IEC standards (Garcia, 2005). The reader is referred to www.pmi.org for more information regarding the work in progress by the PMI (PMBOK, 2004). Each of the standards focuses on a specific process. The ISO 9000 is a quality system standard developed by ISO, and comprises a continuous cycle of planning, controlling and documenting quality in an organisation (Marchewka, 2003; Schwalbe, 2006). The ISO 9000:2000 standard was revised in 2000 and consequently ISO 9001:2000 and ISO 9001:2004 appeared. ISO 9000:2000 describes the fundamental features for a quality management system (QMS), while ISO 9001:2000 illustrates how a QMS can be applied to the creation of products and service provision, and ISO 9001:2004 applies to process management (Hughes and Cotterell, 2006). Another standard that specifically targets software development and contains guidelines for the application of ISO 9001:2000 to computer software is ISO/IEC 90003:2004, entitled "Software Engineering".

In 2006 ISO's portfolio comprised more than 15 900 standards in an effort to provide practical solutions and achieve benefits for almost every sector of economic activity and technology (Nielsen, 2006). According to the Nielsen market research report (2005) 700 000 ISO **9001:2000 certificates** were issued worldwide between 2001 and 2005. In South Africa 5 963 certificates were issued to South African organisations between 2001 and 2005. The ISO 9001:2000 standard will be replaced in Autumn 2008. The ISO standards organisation is thus widely known and accepted as a market standard.

Standards for *software project management* are currently being set. In 2003 a standard was created to target quality management for projects specifically, namely the ISO 10006:2003, which is based on the ISO 9000 and aims to align ISO

10006:2003 with the ISO 9000 family of international standards. The ISO 10006:2003 standard is selected as verification tool. It is based on the well-known ISO 9001:2000 standard. However, ISO 10006:2003 represents a newer standard that targets quality management *in projects* specifically and does not apply to all industrial areas of development, as did ISO 9001:2000. This standard (ISO 10006:2003) concerns the practices required to implement a quality management system in projects specifically, whereas the ISO 9001:2000 targets *all* areas and not software projects development specifically. A work group is currently working towards a South African standard, which is not available yet. As the ISO 9001:2000 is based on the PDCA cycle (ISO 9001:2000; 2000), both the PDCA cycle and ISO 10006:2003 standard are used as verification tools.

9.3 SPMSA MODEL VERIFICATION

9.3.1 PDCA cycle

A specific process, namely the PDCA (Plan-Do-Check-Act) cycle, is known as the operating principle of ISO's management system standards (ISO 9001:2000; 2000). The PDCA cycle was used as the basic approach for developing and improving a certain organisation's management system. This cycle was originally designed by Walther Shewart, but was later revised by the Quality Management authority W. Edwards Deming. It is currently known as the Plan-Do-Check-Act standard, although it has also been referred to as the Shewart cycle and the Deming cycle (Tague, 2004; American Society for Quality, 2005). The cycle is used to coordinate continuous improvement efforts, and it supports daily routine management as well as general problem-solving processes. It furthermore supports SPM, vendor management, human resource management and product development. The PDCA cycle is consequently regarded as suitable for managing the development of software projects.

The basic phases of the PDCA cycle are illustrated in Figure 9.1.

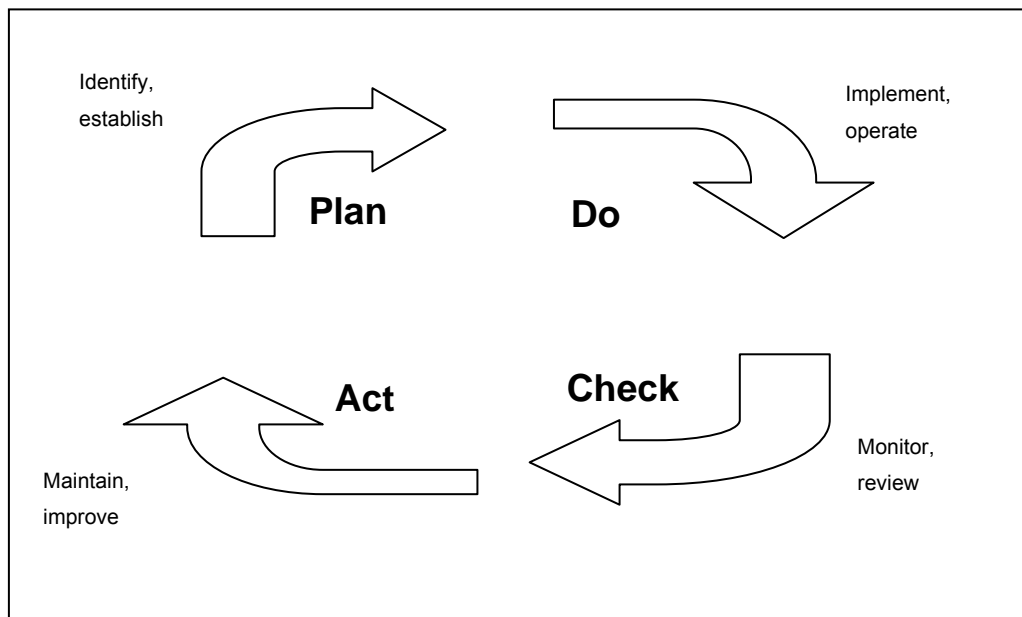


Figure 9.1 The PDCA cycle

The specific phases of this cycle are *plan*, *do*, *check* and *act*. The **plan** phase comprises *identifying and establishing* objectives and processes necessary to deliver results. This includes analysing the organisation's situation, establishing overall objectives, setting interim targets and developing plans to achieve them. Olson (2002) describes it as adhering to the company mission and vision, and basic management objectives based on information gathered through the quality system.

The **do** phase entails *implementing* these plans and processes in order to produce the product or deliver the service in accordance with the customer's requirements (Olson, 2004). The goals and set objectives are thus implemented and produced by **operating** the project-specific products and/or services. The system will thus operate and execute in this phase.

The **check** phase implies the *measurement and monitoring* of this system, which takes place according to information obtained from the customer and the organisation. Actions, documents, deliverables and objectives are *reviewed* in order to improve the system and provide customer satisfaction.

The **act** phase suggests the actions to continually *improve* the plans and processes for implementing the system. Successful activities are *maintained* and less successful ones are improved. This is regarded the key to continual improvement.

As international state of the art, the PDCA cycle is used as the basis for several ISO management system standards, for instance the ISO 9001:2000 standard. For further information concerning the correlation between ISO 9001:2000 and the PDCA cycle, the reader is referred to Brewer and Nash (2005). In Table 9.1. the SPMSA model is firstly verified by drawing a comparison between the SPM phases that form the basis of the model (as depicted in Table 3.2, Chapter 3) and the PDCA cycle.

Table 9.1 PDCA cycle vs SPMSA model

PDCA Cycle	Key function areas of SPM							
	Scope Management	Time Management	Cost Management	Quality Management	HR Management	Communication Management	Risk Management	Procurement Management
	Initiation	Activity definition				Identification and planning	Risk identification	Procurement planning
Plan: Identify Establish	Planning	Activity sequencing Duration estimation	Resource planning	Planning	Organisational planning	Team support	Risk analysis and prioritisation	Solicitation planning
Do: Implement Operate	Definition	Time schedule development	Cost estimation	Assurance	Team development staff acquisition	Information Distribution	Risk Management Planning	Solicitation and source selection
Check: Monitor Review	Verification	Time schedule control	Cost budgeting	Control	Management: Monitor and control	Performance Reporting	Monitor	Contract administration
Act: Maintain, improve	Change control		Monitor, control			Admin closure	Resolution	Contract closure

In the above table, the correlating SPM phases of the SPMSA model are placed alternatively in red (if they correlate with the *plan* phase of the PDCA cycle), green if

they correlate with the *do* phase of the PDCA cycle), blue (if they correlate with the *check* phase of the PDCA cycle) and turquoise (if they correlate with the *act* phase of the PDCA cycle). From Table 9.1 it is clear that the basic phases of the PDCA cycle are clearly represented in the SPMSA model.

Table 9.1 can be generalised by comparing the PDCA cycle to the *generic* phases of software development as reflected in the SPMSA model (compare with Figure 6.2, Chapter 6). This comparison is illustrated in Table 9.2 below. The colour coding is similar to the colour coding of Table 9.1.

Table 9.2 PDCA cycle vs generic phases of the SPMSA model

PDCA cycle	Generic phases of software development
-	Identification / initiation, definition of key functions
Plan	Planning for concepts
Do	Analysis, assessment and evaluation of key concepts
Check	Monitoring, control and management
Act	-

The generic phases of software development as reflected in the SPMSA model also correlate well with the PDCA cycle. The first three phases of the PDCA cycle are similar to the last three phases of the SPMSA model.

9.3.2 ISO 10006:2003

To verify the SPMSA model further, the SPM phases of the SPMSA model (as graphically illustrated in Table 3.2) are compared to the processes in the ISO 10006:2003 standard in Table 9.3. This comparison is made to determine the relevance of the SPMSA model with regard to software project management processes. Thus the SPMSA model is verified against the ISO 10006:2003 standard, which targets projects specifically – see Table 9.3.

Table 9.3 SPMSA model vs ISO 10006:2003

Colour coding: all processes on:					
Quality management: red		Scope management : turquoise		Communication management grey	
Review evaluations: green		Time management: light blue		Risk management: olive green	
Resource management : blue		Cost management: orange		Purchasing management: pink	
ISO 1006:2003 Processes				SPMSA Model	
Non-process clause					
4 Quality management systems	4.1 Project characteristics	4.1.2 4.1.3 4.1.4	Organisations Processes in projects Project management processes	Quality management	None None None
	4.2 Quality management systems	4.2.1 4.2.2 4.2.3	Quality management principles Project quality management Quality plan for project	None None	Quality assurance Quality control Quality planning
Processes					
5.Management responsibility	5.2 Strategic process 5.3 Reviews and progress evaluations				None Performance reporting (communication mngment)
6. Resource management	6.1 Resource-related processes	6.1.2 6.1.3	Resource planning Resource control	Human resource management	Resource planning (cost management) Management: monitor and control
	6.2 Personnel-related processes	6.2.2 6.2.3 6.2.4	Establish project organisational structure Allocation of personnel Team development		Organisational planning Staff acquisition Team development
7.Product realisation	7.2 Interdependency-related processes	7.2.2 7.2.3 7.2.4 7.2.5	Project initiation and management plan development Interaction management Change management Process and project closure		None None Change control (scope) Admin closure (communication)

	7.3 Scope-related processes	7.3.2 7.3.3 7.3.4 7.3.5	Concept development Scope development and control Definition of activities None Control of activities	Scope Management	Initiation Planning Definition Verification Change control
	7.4 Time-related processes	7.4.2 7.4.3 7.4.4 7.4.5	Planning of activity dependencies Estimation of duration Schedule development Schedule control	Time Management	Activity definition and sequencing Duration estimation Time schedule development Time schedule control None
	7.5 Cost-related processes	7.5.2 7.5.3 7.5.4	Cost estimation Budgeting Cost control	Cost Management	Cost estimation Cost budgeting Monitor and control
	7.6 Communication-related processes	7.6.2 7.6.3 7.6.4	Communication planning None Information management Communication control None None	Communication Management	Identification and planning Team support Information distribution None Performance reporting (5) Admin closure (7.2.5)
	7.7 Risk-related processes	7.7.2 7.7.3 7.7.4 7.7.5	Risk identification Risk assessment Risk treatment Risk control None	Risk Management	Risk identification Risk analysis and prioritisation Risk planning Monitor Resolution
	7.8 Purchasing-related processes	7.8.2 7.8.3 7.8.4 7.8.5 7.8.6	Purchasing planning and control Documentation of purchasing requirements Supplier evaluation Contracting Contract control	Procurement Management	Procurement planning Solicitation planning Solicitation and Source selection Contract administration Contract closure
8 Measurement analysis	8.1 Improvement processes	8.1	Improvement		None
	8.2 Measurement and analysis	8.2	Measurement and analysis		None

The ISO 10006:2003 standard consists of a full and extensive list of clauses. Similar to ISO 9001:2000, the ISO 10006:2003 standard consists of eight main clauses, 27 sub-clauses and 61 sub-sub clauses. Main clauses one to three concern only descriptive background, such as the scope of the document, normative references (which state its correlation with ISO 9001:2000), as well as terms and definitions. These clauses are omitted from Table 9.3, as they contain information and not processes to implement. In the standard, each first sub-clause, i.e. 1.1.1, 1.2.1 and 1.3.1 is a general clause, which is also omitted.

In Table 9.3, the correlating phases are placed in a specific similar colour code, e.g. if the SPMSA model's phase correlates with clause 4: Quality management of ISO 10006:2003, both the SPMSA model's phase and the ISO clause are placed in red. Each similar set of processes is colour coded. The colour code appears at the top of Table 9.3.

All the key areas in the SPMSA model, namely scope management, time management, cost management, quality management, human resource management, communication management, risk management and procurement management are reflected in the standard.

Table 9.4 lists the clauses in ISO 10006:2003 that are *not* reflected in the SPMSA model.

Table 9.4 ISO 10006:2003 clauses not reflected in the SPMSA model

<i>ISO 10006:2003 clauses with no equivalent in SPMSA model</i>		
4 Quality management	4.1 Project characteristics	4.2.1 Principles
5 Management responsibility	5.2 Strategic process	
7 Project	7.2 Interdependency-	7.2.2 Project initiation, management

ISO 10006:2003 clauses with no equivalent in SPMSA model		
realisation	related processes	plan 7.2.3 Interaction management
		7.3.3 Scope development and control
		7.6.4 Communication control
		7.7.4 Risk treatment
		7.8.3 Documentation of purchasing requirements 7.8.6 Contract control
	8.1 Improvement processes 8.2 Measurement analysis 8.3 Continual improvement	8.3.1 Continual improvement

From Table 9.4 it is clear that ISO 10006:2003 additionally lists Quality Management (clause 4), Management responsibility (clause 5), Product realisation (clause 7) and Measurement analysis and improvement (clause 8) as separate clauses.

Table 9.5 lists the processes in SPMSA that are **not** reflected in ISO 10006:2003.

Table 9.5 SPMSA processes not reflected in ISO 10006:2003

SPMSA processes not reflected in ISO 10006:2003	
Quality management	Quality assurance Quality control
Scope management	Scope planning Scope verification
Communication management	Communication identification Team support

<i>SPMSA processes not reflected in ISO 10006:2003</i>	
Risk management	Risk planning Risk resolution
Procurement management	Solicitation planning Contract closure

Table 9.5 illustrates that the SPMSA model contains ten processes that are not reflected in the ISO model. As shown in Table 9.3, the majority of processes in the SPMSA model correlate with processes in ISO 10006:2003. Furthermore, the eight core and facilitating functions of the SPMSA model are reflected in the ISO 10006:2003 standard.

The above comparisons clearly indicate that the phases of the SPMSA model conform to the ISO 10006:2003 standard. The SPMSA model is therefore substantiated as conforming to a recognised ISO standard and as such can be justified to be applied to the software project management area. The SPMSA model addresses the shortcomings in current SPM applications and the underlying technology, namely agent technology, will support the unique nature and changing environment of SPM.

9.4 CONCLUSION

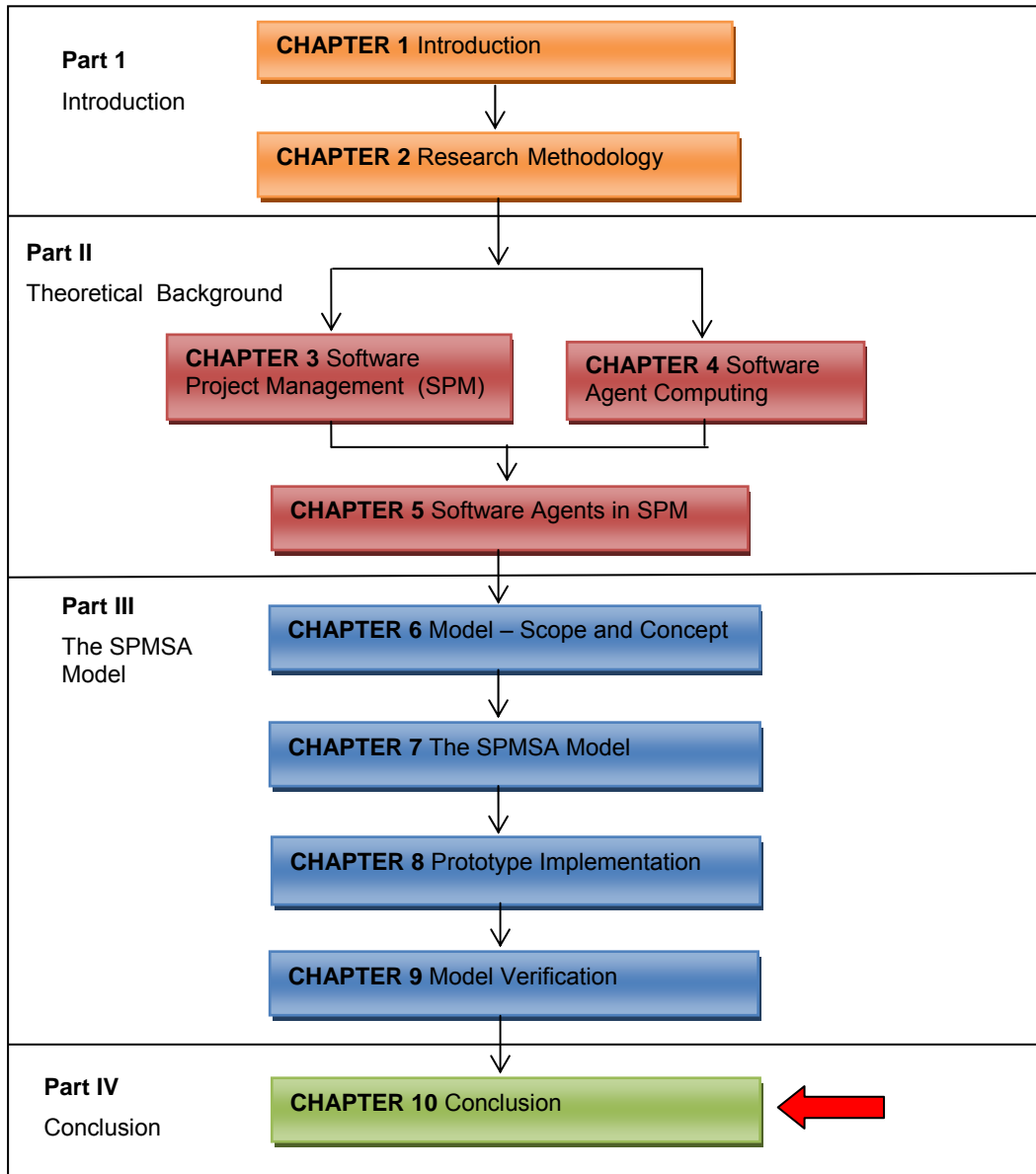
Chapter 9 concludes Part III of this thesis. Part III was devoted to compiling the SPMSA model. As 'proof of concept', a prototype based on a section of the model was designed and implemented. Thereafter the model was verified against the PDCA cycle, as well as against the ISO 10006:2003 standard. Since the PDCA cycle is regarded as the basis of ISO 9000:2001, the SPMSA model was compared to the PDCA cycle with good results. Furthermore, the SPMSA model was compared to the processes prescribed in the ISO 10006:2003 standard, with excellent results. It can consequently be concluded that the SPMSA model complies with industry standards and can be adopted – with minor changes – in the SPM area.

PART IV

CONCLUSION

CHAPTER 10

10 CONCLUSION



10.1 INTRODUCTION

The aim of this concluding chapter is to reflect on the objectives of the present study and to determine the extent to which these objectives have been met. The research questions will therefore be re-examined to ascertain whether they have indeed been answered. Finally, areas of further research will be identified.

10.2 RESEARCH OVERVIEW

This research study explored the processes and procedures associated with *software project management* (SPM) and the ensuing thesis is aimed at making a contribution to enhancing SPM. Problems and challenges in this area were identified and a solution was sought by exploring software agent technology as a new paradigm for supporting SPM processes. An SPM model was accordingly compiled, which enhances SPM processes by incorporating a software agent technology framework to address the shortcomings in this area.

Although the research in hand is aimed at software practitioners and software developers, it will also be beneficial to researchers working in the field of SPM. The development of software projects that support crucial business activities may serve to attain a competitive advantage for an organisation. The quality of the software development process determines the success or failure of many business solutions. Thus, the quality of the software development process, as well as improvements in the development of project management software can result in a significant improvement in software quality (Schwalbe, 2006).

As indicated above, the main issues to be addressed in this study (as reflected in Chapter 1) will be re-examined to determine the extent to which they have been resolved.

10.2.1 Do standard SPM practices take into account the unique nature and changing environment of software projects (SP)?

In the same way that business organisations have grown and evolved over time, computer technology has evolved – yet far more radically. Hence, the science of software project management has had to change and adapt. The discipline of software project management has grown, and together with it, standards, methodologies, best practices and bodies of knowledge. Despite this, software project failures are still common. The increased level of interconnectivity, distribution and processing obviously creates vast challenges in the SPM arena.

The wide range of application areas of software projects and various technologies contributes to the challenges and problems of managing these systems. The area of SPM has changed due to several factors, such as globalisation, advances in computing technology, outsourcing and virtual distributed teamwork. It is clear from many failed or non-satisfactory projects that standard SPM practices fail to support the changes.

Standard SPM practices also fail to address the diverse, unique and changing nature of SPM. Characteristics that are unique to software projects are their invisibility, complexity, conformity and flexibility. These aspects contribute to the difficulty in clearly pinpointing a software project as an exact task with a specific beginning, an end and deliverables.

The dynamically changing environment of SPM further adds to the complexity of these systems, resulting in higher levels of interconnectivity, higher levels of sharing data and knowledge, task tracking and monitoring. These issues should be optimally supported by SPM processes so as to enable project managers to concentrate on crucial issues and strive for lower failure and higher success rates in software projects.

Development methods should thus take full cognisance of this unique nature and changing environment of SPM. It can be concluded that traditional SPM methods do not address the added complexities found in an ever-evolving distributed environment.

10.2.2 How can SPM processes be supported and enhanced in a distributed environment?

The processes and functions in the SPM environment were explored and discussed to delineate various SPM processes and functions. The aim was to get a clear picture of the entire SPM process in an effort to determine the aspects that cause project failure. In order to determine how SPM processes can be enhanced and supported, these processes and functions must be clearly understood.

A paradigm is needed that will support the unique nature and dynamic changing environment of the SPM arena. For this reason software agent computing was investigated. Software agents are adaptive, flexible, pro-active, reactive, and have a collaborative, social nature. Furthermore, software agent technology can address developments such as changing environments, e-business and Internet applications, varying team structures, and open and dynamic environments consisting of heterogeneous components that must interact and span organisational boundaries. These aspects, for instance dynamic, open and distributed environments, are typical features of the software project management arena. Agent behaviour can furthermore be used to support individual team members in numerous tasks, such as coordination and cooperation with team members, document retrieval and distribution, workflow monitoring and control, scheduling and organising meetings, reminders for tasks and overdue dates or deliverables.

It can be concluded that software agent technology is suitable for addressing the various unique features of SPM. Software agent technology provides a suitable framework for supporting and enhancing SPM processes in a complex distributed environment. Flexible management in an ever-changing organisational structure

such as dealt with in SPM is suitably addressed by the computational mechanism of agent systems.

10.2.3 Has software agent technology been applied to the SPM environment?

Agent technology has been applied to various development areas, such as network and system management, decision and logic support, interest matching, data collection in distributed and heterogeneous environments, searching and filtering, negotiating, and monitoring. Although agent technology has indeed been applied to the SPM environment, it has not been applied to the whole spectrum, i.e. *to all core and facilitating functions of SPM*. Agents in SPM applications are typically constrained to one or two of the core and facilitating functions, such as planning, scheduling, human resource management or communication. This is an unfortunate limitation of current software agent applications. Supporting and enhancing the whole spectrum of SPM processes by software agents could provide software project managers with significant advantages over contemporary methods. For example, the coordination and cooperation of teams that integrate the whole spectrum of functionality of a distributed project will be supported and enhanced by the agent features highlighted in this thesis.

10.2.4 How can software agent technology be incorporated and utilised by SPM to enhance the entire SPM environment?

It has been established that agent technology is suitable to supporting SPM. The key areas of SPM were explored and the phases adapted to compile a comprehensive model of SPM functionality to be supported by software agent technology. The model, entitled the SPMSA (Software Project Management supported by Software Agents) model was developed, and it enhances and supports *all* core and facilitating functions of SPM through utilising an agent framework. The SPMSA model consequently addresses the entire spectrum of software project management. It is unique as it supports *each* key function of SPM with a *team of*

software agents. This model is thus specifically tailored to provide support for the constantly changing environment and the unique features of SPM.

A prototype (JPMPS) of a section of the SPMSA model was implemented as ‘proof of concept’ and was tested in a true-life project. The SPMSA model was furthermore verified against the PDCA cycle, as well as against the ISO 10006:2003 standard to substantiate its relevance. These comparisons reflected favourably on the SPMSA model, and it was concluded that the SPMSA model would be suitable to support and enhance the entire SPM environment.

10.3 CONTRIBUTION OF SPMSA MODEL

The SPMSA model was compiled to enhance standard SPM practices and address challenges encountered due to the unique and changing environment of SPM.

Software project management is characterised by *invisibility, complexity, conformity and flexibility*. The SPMSA model is specifically tailored to address each of these unique features through the agent framework. As is evident from work presented in this thesis, agent technology is extremely suitable to handle complex and dynamically *changing environments*. Furthermore, it limits the impact of the invisibility of projects through agent support. In other words, by continuous prompting for task status to identify risks, it supports the management of tasks as a whole. Additionally, agents will provide continuous support to address the distributed SPM environment by executing asynchronously and autonomously, thus lessening network load and communication overhead. The software agent environment will support dynamic changing and relocation of team members, as found in a distributed project.

A software agent framework will improve *automated control* and support *human interaction* by automating workflow management and process coordination. Team interaction will be supported by agents that interact towards a similar goal, supporting coordination and collaboration.

Furthermore, agents will excellently address and manage *complex tasks* through the automation of calculations, dynamic reporting, and the continuous monitoring of maintenance, progress status, as well as risks. Agents can also provide *intelligent support* for software projects (e.g. acting as bidding and negotiating agents) and through dynamic resource allocation. Individual team members can also be supported by a personal assistant agent for each team member.

Finally, the SPMSA model supports and enhances the *entire* environment of the SPM arena. Each key feature of SPM is supported by a team of software agents.

10.4 FUTURE RESEARCH

It has to be conceded that there are limitations and challenges to the model, technology and the prototype. Limitations that were identified can however be explored and will open up new areas for further research. The following limitations and areas of further research exist:

The possibilities of the *model's* interaction with current project management application packages have not been explored in this study. Further research could test the model to tried and tested SPM applications.

The knowledge areas of SPM as depicted in Table 3.2 have been compiled by the researcher. There are a few minor differences between this representation and that of PMBOK (2004). The project management integration function may be regarded as separate function as prescribed by PMBOK (2004). Additionally administrative closure could be included in the integration management function. Table 3.2 and Table 9.1 may be expanded to include this function to be supported by an agent framework. This may be addressed in further research.

The current research does not entail a study on the most effective use of *agent technology*. The agent design will determine the efficiency and effectiveness of the system. Too large an amount of code or functionality within one agent might reflect

negatively on the system. Thus, another area for future research will be to implement more functions and test the trade-off between thin agents with less functionality or larger agents with more functionality.

Agent analysis and design methodologies are being developed but are still without a set standard for development methods and techniques. No standard method is available for the analysis and design of agent systems. This could lead to interesting further research. AML (extension of UML) has been developed and could be tested for diagramming efficiency. Standards are limited and uniform development methods scarce.

The *prototype* in this research study implements only a section of one of the SPM processes. The SPMSA model covers the entire SPM spectrum. Thus, as further research, the prototype could be expanded to implement the entire SPMSA model.

The prototype furthermore implements the section of the SPMSA model in one agent platform, namely JADE based on JAVA. This could also be implemented in other environments, such as C# or agent platforms to test effectiveness, robustness and scalability. Researchers are currently involved in exploring various trade-offs of agent design.

11 REFERENCES

- ADDISON, T. and VALLABH, S. 2002. Controlling Software Project Risks. In: *Proceedings of SAICSIT, South African Institute for Computer Science and Information Technology*. Johannesburg. 128-140.
- ALTMANN, J., GRUBER, F., KLUG, L., STOCKNER, W and WEIPPL, E. 2000. Using mobile agents in the real world: A survey and evaluation of agent platforms. *Software Competence Centre Hagenberg*. URL: www.iwiswn.usv.ro/representations/mobile_agents_GP.pdf. Accessed 12/05/2005.
- AMERICAN SOCIETY FOR QUALITY. 2005. *Project Planning and Implementing Tools*. ASQ. URL: www.asq.org. Accessed 20/10/2006.
- ARIDOR, Y. and LANGE, D.B. 1998. Agent Design Patterns: Elements of agent application design. In: *Proceedings of 2nd International conference on Autonomous Agents*. Minneapolis/St Paul, USA. 108-115.
- ARIDOR, Y. and OSHIMA, M. 1998. Infrastructure for Mobile agents: Requirements and design. In: *Proceedings of the first International Conference on Mobile Agents '98*. Los Alamitos. LNCS **1477**, 38-49.
- BADICA, C., POPESCU, E., FRACKOWIAK, G., GANZHA, M., PAPRZYCKI, M., SZYMCZAK, M. and PARK, M. 2008. On Human Resource Adaptability in an Agent-Based Virtual Organization. In: *Conference on Adaptive Networked Systems and Media, ANSYM 2008*. Wroclaw, Poland.
- BALASUBRAMANIAN, S., BRENNAN, R.W. and NORRIE, D.H. 2001. An Architecture for metamorphic control of holonic manufacturing systems. *Computers in Industry*, **46**, 13-31.
- BELLIFEMINE, F., CAIRE, G., POGGI, A. and RIMASSA, G. 2003. *JADE: A White Paper*, exp, **3(3)**, 6-19.
- BEN-ARI, M. 1990. *Principles of Concurrent and Distributed Programming*. New Jersey: Prentice-Hall. 350. ISBN 0-1371-1821x.
- BENFIELD, S., HENDRICKSON, J. and GALANTI, D. 2006. Making a strong business case for Multiagent Technology. In: *Proceedings of the AAMAS '06*. 1-59593-303 ed. Hokkaido, Japan, ACM Computing surveys. 8-12.
- BERNERS-LEE, T., HENDLER, J. and LASSILA, O. 2001. The semantic web. *The Scientific American*, **5 (1)**, URL: <http://www.scientificamerican.com/2001/0501issue/0501berners-lee.html>. Accessed 20/9/2006.
- BOEHM, B.W. 1991. *Software Risk Management: Principles and Practices*. *IEEE*

- Software*, **8** (1), 32-41.
- BOOTH, A. 2004. *Counting what counts*. URL: <http://www.northumbria.ac.uk/static/powerpoint/Booth.ppt>. Accessed 10/05/2005.
- BORDINI, R., DASTANI, M., DIX, J. AND SEGHROUCHNI, A. (EDS). 2005. Programming Multi-Agent Systems. *Proceedings of the Third international Workshop, ProMAS 2005*. Utrecht, The Netherlands, July 2005. LNAI 3862. Springer. 266.
- BRAUN, P., EISMANN, J., ERFURTH, C. and ROSSAK, W. 2001. TRACY: A Prototype of an Architected Middleware to support Mobile Agents. In: *Proceedings of the Eighth Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems (ECBS'01)*. Washington, D.C. USA. Washington, D.C. IEEE Computer Society. **6**. ISBN 0-7695-1086-8.
- BREWER, D. and NASH, M. 2005. The similarity between ISO 9001 and BS 7799-2. *Gamma Secure Systems*. UK. 1-4 URL: <http://www.gammasl.co.uk/topics/ics/9001Similarities.pdf>. Accessed 20/10/2007.
- BROOKS, F.P. 1979. *The Mythical man-month: Essays on software engineering*. 3rd Edition. Reading, Massachusetts: Addison Wesley Publishing Company. ISBN 0-201-00650-2.
- BROOKS, F.P. 1987. No Silver Bullet: Essence and Accidents of Software Engineering. *Computer*, **20**, (4), 10-19.
- BURMEISTER, B. 1996. Models and Methodology for Agent-Oriented Analysis and Design. Fischer, K. (Ed.). In: *Working Notes of the KI '96 Workshop on Agent-Oriented Programming and Distributed Systems*. DFKI Document D-96-06.
- BURSTEIN, MCDERMOTT, SMITH, E. and WESTFOLD, S. 2000. Derivation of Glue Code for Agent Interoperation. In: *Proceedings of the 4th International Conference on Autonomous Agents*. Barcelona, Spain. 277-284.
- CALLEGARI, D.A. and BASTOS, R.M. 2007. Project management and software development processes: Integrating RUP and PMBOK. In: *Proceedings of the International conference on Systems Engineering and Modeling*. Herzilya, Israel, IEEE. 1-8.
- CAUDRON, M., GROOTE, J.F., VAN HEE, K., HEMERIK, K., SOMERS, L. and VERHOEFF, T. 2004. *Software Engineering Reference Framework*. URL: <http://www.win.tue.nl/~jfg/articles/CSR-04-39.pdf>. Accessed 20/10/2005.

- CHANDRASHEKAR, S., MAYFIELD, B. and SAMADZADEH, M. 1993. Towards automating software project management. *The International Journal of Project Management*, Elsevier Science Ltd. **11(1)**, 29-39.
- CHANG, C. and CAI, L. 2001. Agent based requirements evolution over the Internet. In: *Proceedings of the 2001 Symposium on Applications and the Internet-Workshops*. San Diego, CA, USA. IEEE. 83.
- CHARETTE, R. 2002. *The state of Risk Management: Hype or Reality?* Arlington, MA US, Cutter Information Corporation.
- CHARETTE, R., ADAMS, K. and WHITE, M. 1997. Managing risk in Software Maintenance. *Software*, **14**, 43-50.
- CHEN, F., NUNAMAKER, J.F., ROMANO, N.C.J. and BRIGGS, R.O. 2003. A Collaborative Project Management Architecture. In: *Proceedings of the 36th Hawaii International Conference on System Sciences*. Big Island, Hawaii. IEEE. 1-10.
- CHEN, K., LIN, T., BLOCKER, E. and COKINS, G. 2005. *Overview of Blocker/Chen/Cokens/Lin, Cost Management: a strategic emphasis*. Third Edition. New Jersey: McGraw-Hill. ISBN 0-0728-18360.
- CHEONG, C. and WINIKOFF, M. 2005. Hermes: Implementing goal-oriented agent interactions. In: BORDINI, R.H., DASTANI, M., DIX, J. and SEGHROUCHNI (Eds), *Proceedings of the Third International workshop ProMAS 2005*. 0302-9743 ed. Utrecht, Netherlands: Springer-Verlag. LNAI 3862 (1), 168-183.
- CHMIEL, K., CZECH, D. and PAPRZYCKI, M. 2004a. Agent technology in modelling E-Commerce Processes, Sample implementations. *Multimedia and network Information systems*, **2**, 13-22.
- CHMIEL, K., TOMIAK, D., GAWINECKI, M., KARCZMAREK, P., SCYMCZAK, M. and PAPRZYCKI, M. 2004b. Testing the efficiency of the JADE platform. In: *Proceedings of the ISDP Conference 2004*. Los Angeles: IEEE CS Press, 49-56.
- CLEETUS, K.J., CASCAVAL, G.C. and MATSUZAKI, K. 1996. PACT: A software package to manage projects and coordinate people. In: *Proceedings of the 5th International Workshop on Enabling Technologies: Infrastructure for collaborative Enterprises (WET ICE '96)*. Stanford, CA. USA. IEEE CS Press, 162-169.
- COLLINOT, A., DROGUL, A. and BENHAMOU, P. 1996. An agent oriented design of a soccer robot team. In: *Proceedings of the 2nd International Conference on multi-agent systems*. Kyoto, Japan. 41-47.
- COSENTINO, M., BURRAFATO, P., LOMBARDO, S. and SABATUCCI, L. 2002.

- Introducing Pattern Reuse in the design of Multi-Agent Systems*. URL: <http://citeseer.ist.psu.edu/cache/papers/cs/26501http:zSzzSzwww.csai.unipa.itzSzcossentinozSzpapersZsAITA02.pdf>. Accessed 03/08/04.
- CROFT, D.W 1997. *Intelligent software agents: Definitions and Applications*. URL: <http://www.alumni.caltech.edu/~croft/research/agent/definition>. Accessed 20/10/2005.
- D'Agents: Mobile Agents at Dartmouth College*. 2002. URL: <http://agent.cs.dartmouth.edu/>. Accessed 11/05/2005.
- DALE, J. 1997. *A mobile architecture to support distributed resource information management*. PhD, University of Southampton, Southampton.
- DE VILLIERS, M. 2005. Interpretive research models for informatics: Action Research, Grounded Theory, and the Family of Design and Development research. *Alternation*, **12**, 10-52.
- DEBENHAM, J. and HENDERSON-SELLERS, B. 2003. Designing agent-based systems: Extending the OPEN process framework. In: PLEKHANOVA, V. (Ed.) *Intelligent agent software engineering*. Hershey: Idea group publishing. 1-59140-046-5. (1), 241.
- DEKKERS, C. and FORSELIUS, P. 2007. Increase ICT project success with concrete scope management. MUNCH, J.A. (Ed.) In: *Proceedings of the 8th International Conference on Product-Focused Software Process*. Riga, Latvia, Berlin: Springer-Verlag. LNCS 4589A1. 407-409.
- DORAN, J.E., FRANKLIN, S.R.J.N. and NORMAN, T.J. 1997. On cooperation in Multi-agent systems. *The Knowledge Engineering Review*, **12** (3), 309-314.
- DOWLING, P. and WELCH, D. 2004. *International Human Resource management: managing people in multi-national context*. 4th Edition. Cincinnati: Thompson Learning.
- ELEC 4704. 2003. *Software Project Management*. Department of Electrical and Information Engineering, University of Sydney. URL: <http://www.ee.usyd.edu.au/elec4704/lec-01.html>. Accessed 20/05/2003.
- EVARISTO, R. and VAN FENEMA, P.C. 1999. A typology of project management: emergence and evolution of new forms. *International Journal of Project Management*, **17**, 275-281.
- FERIDUN, M. and KRAUSE, J. 2001. A framework for distributed management with mobile components. *Computer Networks*, **35**, 25-38.
- FIPA. 2003. *The Foundation for Intelligent Agents*. URL: <http://www.fipa.org/specifications/index.html>. Accessed 12/10/2005.

- FLOWER, S. 1996. *Software failure*. New Jersey: Wiley and Sons.
- FRANKLIN, S. and GRAESSER, A. 1996. Is it an Agent, or just a program? A taxonomy for Autonomous Agents. In: *Proceedings of the Third International workshop on Agent Theories, Architectures and Languages*. Berlin: Springer-Verlag. 2135.
- GAETA, M. and RITROVATO, P. 2002. Generalized Environment for Process Management in Cooperative Software Engineering. In: *Proceedings of the 26th Annual International Computer Software and Applications Conference*. Oxford, England. 1-5.
- GANZHA, M., GAWINECKI, M., PAPRZYCKI, M., GASIOROWSKI, R., PISAREK, S. and HYSKA, W. 2006. Utilizing semantic web and software agents in a travel support system. In: A.F. Salam and Stevens (Eds). *Semantic Web Technologies and eBusiness: Virtual Organization and Business Process Automation*. Hershey, USA: Idea Publishing Group. 325-359.
- GAWINECKI, M., KRUSZYK, M., PAPRZYCKI, M., and GANZHA, M. 2007. Pitfalls of agent system development on the basis of a travel support system. In: W. Abramowicz (Ed). *Proceedings of the BIS 2007 Conference*, Berlin: Springer. LNCS 4439, 488-499.
- GARCIA, S. 2005. How standards enable adoption of project management practice. *IEEE Software*, 22-29.
- GELBARD, R., PLISKIN, N. and SPIEGLER, I. 2002. Integrating systems analysis and project management tools. *International Journal of Project Management*, Elsevier Science Ltd. **20** (6), 461-468.
- GIANG, N. and TUNG, D. 2002. Agent platform evaluation and comparison. *Institute of Informatics. Slovak Academy of Science*.
- GREEN, S., HURST, L., NANGLE, B., CUNNINGHAM, P., SOMERS, F. and EVANS, R. 1997. *Software agents: a review*. URL: <http://www.cs.tcd.ie/Brenda.Nangle/iag.html>. Accessed 20/05/2005.
- GREIF, I. 1994. Desktop agents in group-enabled products. *Communications of the ACM*, **37**, 100-105.
- GRIMLEY, M.J. and MONROE, B.D. 1999. Protecting the integrity of agents: an exploration of letting agents loose in an unpredictable world. *ACM Computing surveys*, 1-12.
- GRUBER, T.R. 1993. *Towards principles for the design and ontologies used for knowledge sharing*. KSL -93-04. Stanford University.
- GSCHWIND, T., FERIDUN, M. and PLEISCH, S. 1999. ADK: Building Mobile

- Agents for Network and Systems management from reusable components. In: *Proceedings of the Third International Symposium on Agent Systems and Applications*. Palm Springs, CA. 13-21.
- HALL, G., GUO, Y. and DAVIS, R.A. 2003. Developing a Value-Based Decision-making Model for Inquiring Organizations. In: *Proceedings of the 36th Hawaii International Conference on System Sciences*. Big Island, Hawaii. 111.
- HANSEN, K.T. 2006. Project Vizualization for Software. *IEEE Computer Society*, 84-92.
- HARRISON, C.G., CHESS, D.M. and KERSHENBAUM, A. 1995. *Mobile Agents: Are they a good idea?* URL: <http://216.239.59.104/search?q=cache>. Accessed 12/04/2007.
- HASS, A., JOHANSEN, J. and PRIES-HEJE, J. 1998. Does ISO 9001 increase software development maturity? In: *Proceedings of the 24th EUROMICRO Conference*. Washington, DC. IEEE Computer Society. (2) 860-866.
- HAYES-ROTH, R. and AMOR, D. 2003. *Radical Simplicity transforming Computers Into Me-Centric Appliances*. New Jersey: Prentice Hall.
- HORVAT, H.D.C., MILUTINOVIC, D., KOCOVIC, P. and KOVACEVIC, V. 2000. Mobile Agents and Java Mobile Agent Toolkits. *Telecommunication Systems*, Springer Netherlands. **13** (1-3), 271-287.
- HUGHES, B. and COTTERELL, M. 2006. *Software Project Management*. Fourth Edition. London: McGraw-Hill. ISBN 10-0077-109899. 356.
- HUHNS, M.N. 2002. Agents as Web Services. *IEEE Internet Computing*, 93-95.
- IEEE STANDARDS BOARD 1987. *IEEE Standard for Software Project Management Plans*. IEEE Std. ISBN 0-7381-0409-4, SS 12138.
- IEEE STANDARDS BOARD 1997. *IEEE Standard for Software Project Management Plans*. IEEE Std.
- IGLESIAS, C.A., GARIJO, M., GONZALEZ, J.C. and VELASCO, J.R. 1998. Analysis and design of multi-agent systems using mas-commonKADS. In: *Proceedings of Intelligent Agents IV: Fourth International Workshop on Agent Theories, Architectures and languages*. Rhode Island, LNAI 1365, Springer-Verlag. 313-326.
- IMPEY, R. and FORESTER, G. 2003. *Mobile agent-based enabling technologies for ad-hoc virtual teams and the Internet*. URL: [Http://iit-iti.ncr-cnrc.gc.ca/projects-projects/agent-based-tech-agent-mobiles-e.html](http://iit-iti.ncr-cnrc.gc.ca/projects-projects/agent-based-tech-agent-mobiles-e.html). Accessed 10/04/2007.

- International Standards Organization ISO, 2000. ISO 9001:2000 Quality Management Systems. In: SABS (Ed.) *South African National Standard*. Switzerland, SABS.
- International Standards Organization ISO, 2003. ISO 10006:2003 Quality Management Systems. In: SABS (Ed.) *South African National Standard*, Switzerland, SABS.
- JENNINGS, N.R. 2001. An Agent-Based approach for building Complex Software Systems. *Communications of the ACM*, **44**, 35-39.
- JENNINGS, N.R., SYCARA, K. and WOOLDRIDGE, M. 1998. A roadmap of agent research and development. *Journal of autonomous agents and multi-agent systems*, **1**, 7-38.
- JENNINGS, N.R. and WOOLDRIDGE, M. 1998. *Applications of intelligent agents*. In: Jennings and Wooldridge (Eds). *Agent technology: Foundations, applications and markets*. Springer-Verlag. 3-28.
- JONSSON, N., NOVOSEL, D., LILLIESKOLD, J. and ERIKSSON, M. 2001. Successful Management of Complex, Multinational R and D projects. In: *Proceedings of the thirty-fourth Annual Hawaii International Conference on System Sciences*. Maui, Hawaii, IEEE Computer Society Press. ISBN 0-7695-0981-9. 8044.
- JOSLIN, D. and POOLE, W. 2005. Agent-based simulations for software project planning. In: *Proceedings of the 2005 Winter Simulation Conference*. Orlando. IEEE Computer Society. 8.
- KEFALAS, P., HOLCOMBE, M., ELEFThERAKIS, G. and GHEORGHE, M. 2003. *A Formal Method for the development of Agent-based Systems*. London: Idea Group Publishing.
- KENDALL, E.A., KRISHNA, P.V., SURESH, C.B. and PATHAK, C.V. 2000. An application framework for intelligent and mobile agents. *ACM Computing surveys*, **32**(1).
- KEPHART, J.O. and CHESS, D.M. 2003. The vision of Autonomic computing. *IEEE Computer*, Jan 2003. 41-50.
- KEPHART, J.O. and CHESS, D.M. 2004. PGMS: A P2P-Based Grid Monitoring System, *LNCS 3251/2004*. Berlin:Springer-Verlag. 906.
- KEPHART, J.O. and GREENWALD, A.R. 1999. Shopbot economics. In: *Proceedings of the Fifth European Conference on Symbolic and Quantitative Approaches to Reasoning with Uncertainty*. Seattle. 208-220.
- KETTUNEN, P. and LAANTI, M. 2004. How to steer an embedded software project: tactics for selecting the software process model. *Information and Software*

- Technology*, **47**, 587-608.
- KINNY, D. and GEORGEFF, M. 1997 Modelling and design of multi-agent systems. *Intelligent Agents III*, LNAI **1193**, Springer-Verlag, 1-20.
- KLEIN, H.K. and MYERS, M.D. 1999. A set of principles for conducting and evaluating interpretive field studies in Information Systems. *MIS*, **23**, 67-94.
- KORB, W., ENGEL, D., BOESECKE, R. and EGGERS, G. 2003. Risk analysis for a reliable and safe surgical robot system. *International Congress Series*, **1256**, 766-770.
- KOTZ, D. and GRAY, R. 1999. Mobile agents and the future of the Internet. *Operating systems review*, **33**, 7-13.
- KOTZ, D., JIANG, G., GRAY, R., CYBENKO, G. and PETERSON, R. 2000. Performance Analysis of Mobile Agents for Filtering Data Streams on Wireless Networks. *Workshop on Modeling, Analysis and Simulation of Wireless and Mobile Systems (MSWiM 2000)*. Boston, Massachusetts. ACM Press. 85-94.
- KRUPANSKY, J.W. 2003. *What is a software agent?* URL: <http://activity.com/agdef.htm>. Accessed 12/10/2005.
- KRUSZYK, M., GANZHA, M., GAWINECKI, M. and PAPRZYCKI, M. 2007. Pitfalls of agent system development on the basis of a travel support system. In: *Proceedings of the Business Information Systems BIS 2007 Conference*. Poznan, Poland. LNCS **4439**, Springer-Verlag, 488-499.
- LANGE, D.B. and OSHIMA, M. 1998. *Programming and deploying Java Mobile agents with Aglets*. Reading, Massachusetts: Addison-Wesley. 225.
- LANGE, D.B. and OSHIMA, M. 1999. Seven good reasons for mobile agents. *Communications of the ACM*, **42**, 88-89.
- LETHBRIDGE, T.C. and LAGANIERE, R. 2001. *Object-oriented Software Engineering: Practical Software development using UML and Java*. London: McGraw-Hill.
- LEUNG, H.K.N. and POON, C. 1999. Multi-Agent environment for Quality Assurance, In: *Proceedings of the Euromicro Conference, AUTOQ*. Milan, 2294.
- LIND, J. 2001. Issues in Agent-oriented software engineering. In: PIANCARINI, P. and WOOLDRIDGE, M. (Eds). *Proceedings of the First International workshop AOSE-2000*. Berlin: Springer-Verlag **1957**, 45-58.
- LINGAU, A., DROBNIK, O. and DOMEL, P. 1995. *An HTTP-Based Infrastructure for Mobile Agents*. URL: <http://citeseer.nj.nec.com/lingau95httpbased.html>.

Accessed 09/03/2005.

- LUCK, M. and D'INVERNO, M. 2004. *A Conceptual Framework for Agent Definition and Development*. URL: <http://www.agentlink.org>. Accessed 26/05/2007.
- LUCK, M., MCBURNEY, P., SHEHORY, O., WILLMOTT, S., and AGENTLINK COMMUNITY. 2005. *Agent Technology Roadmap*. URL: <http://www.agentlink.org>. Accessed 25/7/2005.
- MAES, P. 1994. Agents that reduce work and information overload. *Communications of the ACM*, **37**(7) 31-40.
- MARCH, S. and SMITH, G. 1995. Design and natural science research on information technology. *Decision support systems*, **15**, 251-266.
- MARCHEWKA, J.T. 2003. *Information Technology Project Management*. Northern Illinois: Wiley and Sons. 319.
- MAURER, F. 1996. Project Coordination in Design Processes. In: *Proceedings of the 5th International Workshop for enabling Technologies: Infrastructure for Collaborative Enterprises (WET ICE'96)*. Stanford, CA: IEEE Computer Society. 191-198.
- MCBRIDE, T., HENDERSON-SELLERS, B. and ZOWGHI, D. 2004. Project Management Capability Levels: An Empirical Study. In: *Proceedings of the 11th Asia-Pacific Software Engineering Conference (APSEC '04)*. Busan, Korea. The Computer Society. 56-63.
- MCMICHAEL, B. and LOMBARDI, M. 2007. ISO 9001 and Agile Development. In: MAURER, F., DAVIES, R., MELNIK, G. and POLLICE, G. (Eds). *Agile 2007, Proceedings*. 978-0-7695-2872-4 ed. Washington, DC, IEEE Computer SOC. 262-265.
- MEHANDJIEV, N., LAYZELL, P., BRERETON, P., LEWIS, G., MANNION, M. and COALLIER, F. 2002. Thirteen knights and the seven-headed dragon; an Interdisciplinary software engineering framework. In: *Proceedings of the 10th International Workshop on Software Technology and Engineering Practice (STEP '02)*. Washington, DC: IEEE. 46-54.
- MEREDITH, J.R. and MANTEL, S.J. 2002. *Project Management – A Managerial Approach*. Fifth Edition. New York: John Wiley and Sons. 765.
- MICHAEL, K. and JAY, J. 1998. *Developing Intelligent Agents for Distributed Systems*. New York: McGraw-Hill. 385.
- MILES, M.B. and HUBERMAN, A.M. 1994. *Qualitative Data Analysis*. Thousand Oaks, London: SAGE Publications. ISBN 0-8039-4653-8.
- MILOJICIC, D., BREUGST, M., BUSSE, I., CAMPBELL, J., COVACI, S.,

- FRIEDMAN, B., KOSAKA, K., LANGE, D., ONO, K., OSHIMA, M., THAM, C., VIRDHAGRISWARAN, S. and WHITE, J. 1998. MASIF: The OMG Mobile Agent System Interoperability Facility. *Personal Technologies*, **2**, 117-128.
- MOREAU, L., ZAINI, N., CRUICKSHANK, D. and DE ROURE, D. 2003. SoFAR: An agent framework for distributed information management. In: Plekhanova, V. (Ed.). *Intelligent agent software engineering*. London: Idea Group Publishing.
- MOUTON, J. 2001. *How to succeed in your Master's and Doctoral Studies*. Pretoria: Van Schaik Publisher. 280.
- MÜLLER, J.P., BAUER, B. and FRIESE, T. 2004. Programming Software agents as designing executable business processes: a model driven perspective. *Programming Multi-Agents systems*, LNAI **3067**, 49-71.
- MYERS, K., BERRY, P., BLYTHE, J., CONLEY, K., GERVASIO, M., McGUINNESS, D., MORLEY, D., PFEFFER, A., POLLACK, M. and TAMBE, M. 2007. An Intelligent Personal Assistant for Task and Time Management. *Artificial Intelligence Magazine*, **28**, 1-19.
- MYERS, M.D. 2006. Qualitative research in Information Systems. *MIS Quarterly*, **21**, 241-242.
- NIELSEN, A. 2006. The ISO Survey of certifications - 2006. Geneva, Switzerland, ISO Central Secretariat.
- NIENABER, R. and BARNARD, A. 2005. Software Quality management supported by Software Agent Technology. In: *Proceedings of the Informing Science and Information Technology Conference (INSITE 2005)*. Flagstaff, USA. 659-670.
- NIENABER, R.C. and BARNARD, A. 2007. A Generic Agent Technology Framework to Support the Various Software Project Management Processes. In: *Proceedings of the International Conference on Issues in Informing Science and Information Technology (INSITE 2007)*. Ljubljana, Slovenia. ISSN 1555-1245. 108.
- NIENABER, R.C. and CLOETE, E. 2003. A Software Agent Framework for the support of Software Project Management. In: *Proceedings of the International Conference on IT Research in Developing Countries (SAICSIT 2003)*. Midrand, Gauteng. 111-113.
- NIENABER, R.C., SMITH, E., BARNARD, A. and VAN ZYL, T. 2008. Software Agent Technology supporting Risk Management in SPM. In: *Proceedings of the IADIS International Conference on Applied Computing (IADIS2008)*. Algarve, Portugal. ISBN 978-972-8924-522. 373.
- NWANA, H.S. and NDUMU, D.T. 1996a. An Introduction to agent technology. *BT Technology Journal*, **14**, 55-67.

- NWANA, H.S. and NDUMU, D.T. 1996b. Research and development challenges for agent-based systems. *IEEE/BCS Software Engineering Journal*, **144**(1), 2-10.
- OBJECT MANAGEMENT GROUP. (2000). Mobile agent facility specification. URL: <http://www.omg.org>. Accessed 08/12/2004.
- O'CONNOR, R. and GAFFNEY, E. 1998. *A Distributed Framework for implementing a Multi-agent Assistant System*. Dublin, Ireland, Dublin City University. 1-12.
- O'CONNOR, R. and JENKINS, J. 1999. Using Agents for Distributed Software Project Management. In: *8th International Workshop on Enabling Technologies: Infrastructures for Collaborative Enterprises*. Stanford, USA. IEEE Computer Society Press. 54-60.
- O'CONNOR, R. and MOYNIHAN, T. 2000. An Agent Model of Decision Support for Software Project Management. *Advances in Decision Technology and Intelligent Information Systems*, **1**, 26-30.
- OATES, B. 2006. *Researching Information Systems and Computing*. London: Sage Publications Ltd.
- OBUDIYI, J.B., KOCUR, D.J. and WEINSTEIN, S.M. 1997. SAIRE: A scalable agent-based information retrieval system. In: *Proceedings of the First International Conference on Autonomous Agents*. Marina del Rey, USA: ACM Press. 292-299.
- ODELL, J. 2001. Key issues for agent technology. *JOOP*, **13**, 23-27.
- OGHMA: 2003. Types of Software Agents. *Oghma: Open Source*. URL: <http://www.oghma.org/>. Accessed 20/05/2003.
- OLIVIER, M. 2004. *Information Technology Research*. Johannesburg: B and D Printers. 176.
- OLSON, C.E. 2002. The Plan, Do, Check, Act Cycle and ISO 9001:2000. *Systems Quality Consulting*. Alta Loma, Systems Quality Consulting. URL: <http://www.systemsquality.com>. Accessed 13/02/2008.
- OLSON, D.L. 2004. *Information Systems Project Management*. Second Edition. Boston: McGraw-Hill. 308.
- OMICINI, A. 2001. SODA: Societies and Infrastructures in the Design and Analysis of Agent-Based Systems. *Agent-Oriented Software Engineering*, LNCS **1975**, 185-193.
- PAI, W.C., WANG, C.C. and JIANG, D.R. 2000. A Software development model based on quality measurement. In: *Proceedings of the 13th International conference on computer applications in Industry and Engineering*. ICISA

'2000. Seattle. 40-43.

- PARUNAK, H.V.D., BAKER, A.D. and CLARK, S.J. 1997. The AARIA Agent architecture: An example of requirements-driven Agent-based system design. In: *Proceedings of the First International Conference on Autonomous agents. ICAA-79*. Marina Del Rey, CA. New York: ACM Press. 482-483.
- PETRIE, C., GOLDMAN, S. and RAQUET, A. 1999. Agent-based project management. *Lecture Notes on Artificial Intelligence (LNAI)*, **1600**, 1-23.
- PHILLIPS, D. 1998. *The Project Manager's handbook*. IEEE Computer Society.
- PINTO, J.K. and SLEVIN, D. 1987. Critical factors in successful project Implementation. *IEEE Transactions on Engineering Management*, **34**, 22-27.
- PROJECT MANAGEMENT INSTITUTE (PMI). 2004. *The Guide to the Project Management Body of Knowledge (PMBOK)*. URL: www.pmi.org. Accessed 10/04/2006.
- PURVIS, R.L., MCCRAY, G.E. and ROBERTS, T.L. 2003. The impact of Project Management Heuristics to IS Projects. In: *Proceedings of the 36th Hawaii International Conference on System Sciences*. Big Island, Hawaii, IEEE. 1-7.
- RIGAUD, E. and GUARNIERI, F. 2002. Towards an Agent Oriented Virtual Organization Dedicated to Risk Prevention in Small and Medium Size Companies. In: *Proceedings of the 13th International Workshop on Database and Expert Systems Applications (DEXA'02)*. IEEE Computer Society.
- ROMANO, N.C., CHEN, F. and NUNAMAKER, J.F. 2002. Collaborative Project Management Software. In: *Proceedings of the 35th Hawaii International Conference on System Sciences*. Big Island, Hawaii. IEEE. 15.
- ROODE, D. 2003. Information Systems Research: A Matter of Choice? *SA Computing Journal*, **30**, 1-2.
- ROSE, J., PEDERSEN, K., HOSBOND, J.H. and KRAEMMERGAARD, P. 2007. Management competences, not tolls or techniques: A grounded examination of software project management at WM-data. *Information and Software Technology*, **49**, 605-24.
- ROTH, V. 2004. Obstacles to the adoption of Mobile Agents. In: *Proceedings of the IEEE International Conference on Mobile Data Management (MDM'04)*. Los Alamitos, CA, IEEE Computer Society. 296-297.
- ROY, G.G. 2004. A Risk Management Framework for Software Engineering Practice. In: *Proceedings of the Australian Software Engineering Conference (ASWEC'04)*. Australia, IEEE Computer Society.
- SAMARAS, G. 2004. Mobile Agents: What about them? Did they deliver what they

- promised? Are they here to stay? In: *Proceedings of the IEEE International Conference on Mobile Data Management (MDM'04)*. Los Alamitos, CA, IEEE Computer Society. 294-295.
- SATZINGER, J. W., JACKSON, R.B. and BURD, S.D. 2004. *Systems Analysis and Design in a changing world*. Boston, Massachusetts: Thompson Course Technology. 779.
- SAUER, J. and APPLERATH, H. 2003. Scheduling the supply chain by teams of agents. In: *Proceedings of the 36th Hawaii International Conference on System Sciences*. Big Island, Hawaii. 81.
- SCHOEMAN, M.A. 2005. *An Approach to facilitating the training of mobile agent programmers and encouraging the progression to an agent-oriented paradigm*. MSc School of Computing. Pretoria, University of South Africa.
- SCHWALBE, K. 2006. *Information Technology Project Management*. Canada: Thomson Course Technology.
- SHEHORY, O., GOLDSTEIN, M., SHULMAN, A., STURM, A. and YUROVITSKY, B. 2002. Bi-concurrent Layered Architecture for eCommerce Agents. In: *Proceedings of the First International Joint Conference on Autonomous Agents and Multi-agent Systems*. Bologna, Italy. 1035-1036.
- SHERER, S.A. 2004. Managing Risk beyond the Control of IS Managers: The Role of Business Management. In: *Proceedings of the 37th Hawaii International Conference on System Sciences*. Big Island, Hawaii. 1-10.
- SMITS, H. and PSHIGODA, G. 2007. Implementing Scum in a distributed software development organization. In: MAURER, D., MELNIK, P (Eds). *Agile 2007*. Washington, DC: IEEE Computer Soc. 371-375.
- SOMMERVILLE, I. and RODDEN, T. 1996. *Human, social and organizational influences of the software process*. Chichester: Wiley. 89-109.
- SONNEKUS, R. and LABUSCHAGNE, L. 2004. Establishing the relationship between IT project management and IT project success in a South African context. In: *Proceedings of the Global Knowledge for Project Management Professionals: PMSA International Conference*. Johannesburg, South Africa. 183-192.
- SUNDSTED, T. 1998. An Introduction to agents. *JavaWorld*. URL: <http://www.javaworld.com/javaworld/jw-06-1998/jw-06-howto.html>. Accessed 03/05/2004.
- SZEJKO, S. 2002. Requirements Driven Quality Control. In: *Proceedings of the Computer Software and Applications Conference*. Oxford, England. 125-130.

- SZYMCZAK, M., FRACKOWIAK, G., GANZHA, M., GAWINECKI, M., PAPRZYCKI, M. and PARK, J. 2007. Resource Management in an Agent-based Virtual Organization. *MaSeB Workshop*. Los Alamitos, CA: IEEE CS Press. 458-462.
- TAGUE, N.R. 2004. Project Planning and Implementing Tools. *The Quality Toolbox*. Second edition, ASQ Quality Press. 390-392.
- TAHARA, Y., OHSUGA, A. and HONIDEN, S. 1999. Agent System Development Method Based on Agent Patterns. In: *Proceedings of the 21st International Conference on Software Engineering*. 356-367.
- TANAIR, D. 2005. Editorial. *Journal of Mobile Information Systems*, **1**, 1-2.
- THAMHAIN, H.J. 2003. Project Management Minitrack: Levering Information Technology. In: *Proceedings of the 36th Hawaii International Conference on System Sciences*. Big Island, Hawaii, IEEE Computer Society. 248.
- THE STANDISH GROUP INTERNATIONAL. 1995. The Chaos Report. In: JOHNSON, J. (Ed.). *Application Development Trends*. URL: www.standishgroup.com. Accessed 06/06/2006.
- THE STANDISH GROUP INTERNATIONAL. 2001. *Xtreme Chaos*. URL: www.standishgroup.com. Accessed 06/03/2006.
- THE STANDISH GROUP INTERNATIONAL. 2003. *Latest Standish Group Chaos Report*. URL: <http://www.standishgroup.com>. Accessed 4/03/2005.
- THE STANDISH GROUP INTERNATIONAL. 2005. Latest Standish Group Chaos Report. *Chaos Chronicles*. Massachusetts. URL: <http://www.standishgroup.com>. Accessed 04/03/2005.
- VAN ZYL, T.L. 2005. *Integrating Secure Resource Negotiating Agents into Telemanufacturing*. MSc. University of Johannesburg.
- VENNERS, B. 1997. *Under the hood: Architecture of aglets*. URL: http://www.javaworld.com/jw-04-1997/jw-04-hood_p.html. Accessed 23/11/2004.
- VERNER, J., OVERMYER, S. and MCCAIN, K. 1999. In the 25 years since The Mythical Man-Month, what have we learned about project management? *Information and Software Technology*, **41**, 1021-1026.
- VERNER, J.M. and CERPA, N. 2005. Australian Software Development: What Software Project Management Practices Lead to Success? In: *Proceedings of the 2005 Australian Software Engineering Conference (ASWEG)*. Austria. IEEE Computer Society Press.
- VUKMIROVIC, M., GAWINECKI, M., KOBZDEJ, P., GANZHA, M. and PAPRZYCKI,

- M. 2007. Implementing message exchange between airlines' GDSs and travel systems with ontologically demarcated data. In: *Proceedings of the 29th Conference on ITI*. Berlin, Springer-Verlag.
- WHITE, J. 1995. Telescript technology: The foundation of the electronic market place. *General Magic white paper*.
- WHITTEN, J.L., BENTLEY, L.D. and DITTMAN, K.C. 2004. *Systems Analysis and Design in a changing world*. Boston: McGraw-Hill Irwin.
- WONG, J., HELMER, G., NAGANATHAN, V., POLAVARAPU, S., HONOVAR, V. and MILLER, L. 2001. SMART mobile agent facility. *The Journal of Systems and Software*, **56**, 9-22.
- WOOLDRIDGE, M. 2002. *MultiAgent Systems*. West Sussex, England: John Wiley and Sons, Ltd.
- WOOLDRIDGE, M., JENNINGS, N.R. and KINNY, D. 2000. The GAIA Methodology for Agent-Oriented Analysis and Design. *Autonomous Agents and Multi-Agents Systems*, **3**, 285-312.
- WU, C. and SIMMONS, D.B. 2000. Plan Tracking Knowledge base. In: *Proceedings of the 24th Annual International Computer Software and Applications Conference, COMPSAC '00*. Taipei, Taiwan. IEEE CS Press. 299-304.
- WURMAN, P. 2001. Dynamic pricing in the Virtual marketplace. *IEEE Internet Computing*, **5**, 52-60.
- ZAMBONELLI, F., JENNINGS, N.R., OMICINI, A. and WOOLDRIDGE, M. 2001. Software Engineering with Agents: Pitfalls and Prarfalls. *IEEE Internet Computing*, 20-27.
- ZANONI, R. and AUDY, J.L.N. 2003. Project Management Model for Physically Distributed Software Development Environment. In: *Proceedings of the 36th Hawaii International Conference on System Sciences (HICSS'03)*. Big Island. Hawaii. 249, 1-7.

A. APPENDIX A

Paper presented at SAICSIT2003, The South African International Conference on Computer Science and Information Technology Research in Developing Countries, Fourways, Gauteng, September 2003.

Paper published in the conference proceedings. ACM. ISBN: 1-58113-774-5, 16-23.

A Software Agent Framework for the Support of Software Project Management

RITA NIENABER
 UNIVERSITY OF SOUTH AFRICA
 AND
 ELSABE CLOETE
 UNIVERSITY OF SOUTH AFRICA

Numerous software development projects do not live up to expectations or sadly fail. This can be seen in the fact that software projects often do not comply with the traditional standard measurements of success, namely time, cost and specifications. Traditionally, individual software projects were executed at a single location. However, due to globalisation and advances in computing technologies, this has changed, and software projects are developed and deployed in distributed and collaborative environments. This means that traditional project management methods cannot and do not address the added complexities found in a distributed environment, such as efficient task scheduling, tracking and monitoring, as well as effective sharing of information and knowledge among project contributors. High levels of collaboration, task interdependence and distribution have become essential across time, space and technology. In this paper the utilisation of stationary and mobile software agents is investigated as a potential tool to improve software project management processes. We also propose and discuss a software agent framework to support distributed 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 timeously, within budget and to users' satisfaction

Categories and Subject Descriptors: D1.3 [Programming Techniques]: Concurrent Programming - *Distributed Programming; Concurrent Programming*; K6.1 [Management of Computing and Information Systems]: Project and People Management - *Systems development; Strategic information planning*; K6.3 [Management of Computing and Information Systems]: Software Management - *Software process*.

General Terms: Design, Experimentation, Management

Additional Key Words and Phrases: Software agent computing, Software project management, Collaborative distributed software projects.

1. INTRODUCTION

Software applications are integrated into almost every business application today. It is the quality, effectiveness and efficiency of these applications that determine the success or failure of many business solutions. As a result, businesses often find that they need to attain a competitive advantage through the development of software projects that support crucial business activities. The quality of a software project is determined by the quality of the software development process. Improvements in the development process can result in significant improvement in software quality [Schwalbe 2000].

Over the past few decades, software projects generally failed to come up to user expectations, were commonly delivered late, and mostly ran over the set budget. Indeed, much of this still holds true today, and it has alerted software developers and managers to the fact that these issues have to be addressed in concrete terms, and as a result the field of *Software Project Management (SPM)* has evolved. SPM involves the management of all aspects and issues that are 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 and control, as well as managing contracts, teams of people and quality. Initially, traditional *Project Management (PM)* techniques were applied to the development of software projects. Different standard project management approaches exist, which are applicable to different areas of SPM, such as PRINCE 2, BS 6079. However, over time PM methods seemed to lack in the ability to address the unique characteristics of the software development arena [Hughes and Cotterell 2002]. This led to the development of SPM as an independent application area and field of study.

In order to address the existing shortcomings in managing software projects, practitioners also attempted to apply several Software Engineering principles to different SPM processes [Lethbridge and Laganiere

2000]. They explored standard structured analysis and design methods, and also incorporated object-oriented approaches to overcome the aforementioned shortcomings [Gelbard et al. 2002; Lethbridge and Laganieri 2000]. Yet disappointment remained since many software projects still failed to comply with the triple constraints of scope, time and cost [Oghma:Open Source 2003]. The triple constraints refer to the fact that the failure of software projects can mostly be attributed to the fact that they are not delivered on *time* and do not meet the expectations of the client (*scope*), and as a result have *cost* implications.

The SPM environment is rapidly changing due to globalisation and advances in computing technology. This implies that the traditional single project, which was commonly executed at a single location, has evolved into distributed, collaborative projects. The focus in the SPM processes has clearly shifted from the position that it held two decades ago. Consequently, tools for effective sharing of information and knowledge among project contributors, as well as efficient task scheduling, tracking and monitoring are sorely needed. High levels of collaboration, task interdependence and distribution have become essential across time, space and technology [Chen et al. 2003].

A promising solution to addressing software management problems in a distributed environment is offered by *software agent technology*. 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 team environment, where software agents can monitor and coordinate events and meetings and distribute documentation [Balasubramanian 2001].

SPM skills, especially in the distributed computing environment, are greatly in demand. Moreover, there is a desperate need for technologies and systems to support the 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 to improve the SPM processes. We concern ourselves with the question of how software agents can be used to improve SPM processes in a distributed environment. As a result, we propose a software agent framework to support distributed SPM. Although our research is not yet complete, initial indications are that it will be significant in enabling software developers to meet market expectations, which will bring about savings in cost, time and effort.

Section 2 contains a background study and a discussion on research in this area. Section 3 provides a generic-type framework for one of the key SPM processes. 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. Finally, Section 4 presents a conclusion.

2. BACKGROUND

2.1 Software Agents

A software agent is a computer 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. For example, Krupansky [2003] distinguishes between *ten* different types of software agents, while the Oghma Open Source [2003] web site identifies sixteen different types of software agents. Because 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.

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 Woolridge [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 timeously 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 specialized services to cause a deliberate synergism amongst agents. It is expected of most agents to have a strong collaborative 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 what sequence, in response to the state of its external environment [Pai et al. 2000].

We consider a stationary agent to be 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.

We consider a *mobile agent* to be 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].

2.2 Software Project Management

The IEEE defines SPM 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 these issues in a framework that contains the key elements in the field of SPM. We distinguish between three key elements: project stakeholders, project management knowledge areas, and project management tools and techniques.

The project *stakeholders* are the people involved in all 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 concerned during 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. Stretched across all these knowledge areas are the project management *tool and techniques* (on the right-hand side of the framework diagram). These are used to assist team members and project managers in carrying out their respective tasks.

This framework refers to the SPM of a single project, in which the SPM manager allocates tasks and gives instructions to various role players. However, as mentioned earlier, factors such as business globalisation, rapid technology advancement, and distributed team membership have influenced the SPM environment to such an extent that traditional tools and techniques supporting the key management areas are no longer adequate [Chen et al.2003]. For example, the focus of the communication and cooperating mechanisms has changed. As a result, the communication between various role players is not sufficiently supported in this environment to coordinate and schedule activities, and support document distribution. 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 [Krupansky 2003]. All these depend on having good communication and coordination skills among the stakeholders. Poor, ineffective or untimely communication, contradictions, omissions, failure to notify all of meetings and

decisions, and failure to store information are often cited as reasons for projects failing or running over time. Traditional reporting tools use a simple passive reporting mechanism, which does not provide sufficient reporting support to a collaborative distributed system [Chen et al.2003]. Communication is, however, enhanced and supported by the use of a common repository. A paper-based repository has several disadvantages, such as retrieval delays, lost documentation and error-proneness, but most of all, may result in insufficient project documentation in the distributed environment. Another common problem in communication is that many project processes, contexts, rationales, or artefacts may not be captured at all. An electronic repository might overcome some of these disadvantages.

2.3 Software Agents in SPM

Software agent technology is being explored as a promising way to support and implement complex distributed systems. Using software agents for SPM processes is a new field of research and as such, literature in this regard is not commonly available. However, some work on using agents has been done to address at least some of the aspects of SPM. In this section, the authors briefly consider how agent technology is currently being deployed in SPM by considering some application examples.

The first application that we mention is intended for the broader project management environment, and is not specific to the SPM environment. Nevertheless, we refer to this example as it applies agent technology to *scheduling tasks*, which are common to both environments.

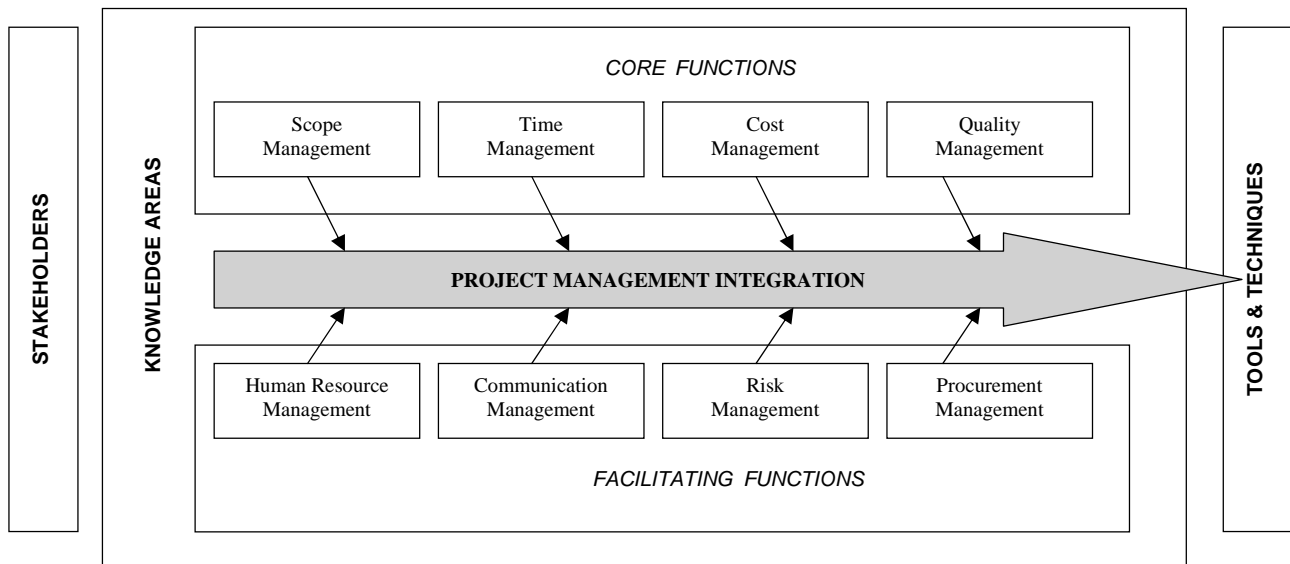


Figure 1: *Software Project Management Framework (adapted from Schwalbe).*

Furthermore, the authors find it worthy of mention it since it is focussed on in the distributed environment. In recent work, Sauer & Applerath [2003] presented an approach that involves using a generic agent framework to support the scheduling tasks within the supply chain in the PM environment. The framework allows for the consistent design of agents that reside on several levels of the organization. To prevent communication overhead (found in earlier multi-agent systems), agent *teams* are formed. All the agents in a team then collaborate to solve a specific scheduling task on a particular level. Furthermore, every agent (in its personal capacity) is also responsible for a specific schedule (the schedule of the resources that it represents). Therefore each agent is provided with the scheduling knowledge that is necessary to create or maintain the schedule without contacting the members of the team.

In another example, Maurer [1996] proposes a system (the *CoMo Kit*) in which methods and tools were developed to plan and manage complex workflows, especially in design domains. According to this system, tasks can be decomposed into subtasks and for every task, several alternative decompositions (methods)

can be defined. Every task is associated with a set of agents, humans or computers, which are able to solve it. The problem-solving process, for example the application of methods to tasks, is distributed via a local area network. The proposed system uses agent technology as a tool for planning, coordinating and designing process execution. This approach follows a centralised black-box agent approach. The system architecture consist of a *modeller*, which does project planning; a *scheduler*, which supports project execution and manages information produced; and an *information assistant* that allows access to the current state of the project. During SPM, the *modeller* gathers information through interaction with the project manager or other stakeholders, and as a result presents a model of this information to the *scheduler* as input. The scheduler then manages agendas that contain the tasks to be carried out by an agent. To work on the task, the agent can access all relevant information (using the information assistant) for solving the problem.

3. SPM FRAMEWORK SUPPORTED BY SOFTWARE AGENT TECHNOLOGY

As described earlier, the software project management 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 exactly 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 et al. 2003].

Before discussing our proposed SPM framework, we briefly reconsider the distinct knowledge areas and practices entailed in software project management (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. The solution presented by Sauer and Apperath [2003] primarily focuses on the Time Management and certain aspects of the Communication Management functions. Maurer's solution [1996] is applicable to the Scope Management, Time Management and to a certain extent the Communication Management 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. For the purpose of this paper, we discuss our approach to only one of the SPM key processes, namely *Communications Management*, and describe the agent framework to accomplish the so called black box for this process.

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 2000]. Effective communication and sharing of information and knowledge among project contributors are needed. Schwalbe [2000] identifies five distinct functions associated with Communications Management, namely (1) communications planning; (2) information distribution; (3) performance reporting; (4) administrative closure; and (5) teamwork support. The *communications planning* function determines the *who*, *when* and *how* of the project, while the *information distribution* function entails disseminating information to keep all the 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.

To describe how software agents can be used to address the different functions of *Communications 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 and Lange 1998; Kendall et al. 2000; Tahara et al.1999] available, accomplish specific agent-associated tasks, such as creation, migration, suspension, and collaboration. The design of the overall system, based on components (specialised agents) simplify the design and programming of agents. The following specialised working agents are used:

- *Messaging agent*: 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 work in a distributed environment.
- *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 needs to sustain its owner. The PA agent is not computer-bound, but human-bound, as its stakeholders may be required to work on different computers when working 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*: an agent responsible for monitoring tasks, reporting back to the communications planning and *information distribution functions* where scheduling and rescheduling of tasks as well as the notification of stakeholders can take place. A monitoring agent is mobile, with intelligence, flexibility and strong collaborative properties.
- *Client agent*: a stationary agent responsible for a specialised task such as information retrieval or gathering. Client agents may or may not have intelligence, depending on their specific task, but must have a collaborative nature to interact with other agents in their agent team.
- *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.

Figure 2 illustrates the main operations in the Management Communications function and how agent teams cooperate to accomplish the objectives of these operations.

The software project manager, or other designated stakeholders, interacts with the communications planning function through a special user interface. This user interface, which sits on top of the *communications planning* function, uses *personal* agents, *task* agents and *messaging* agents. The interface assigns a *personal* agent to the user who has supervision rights over other *personal* agents. During interaction with the interface, the user defines team members or relevant stakeholders as well as the tasks that are assigned to them, and defines milestones, objectives, et cetera.

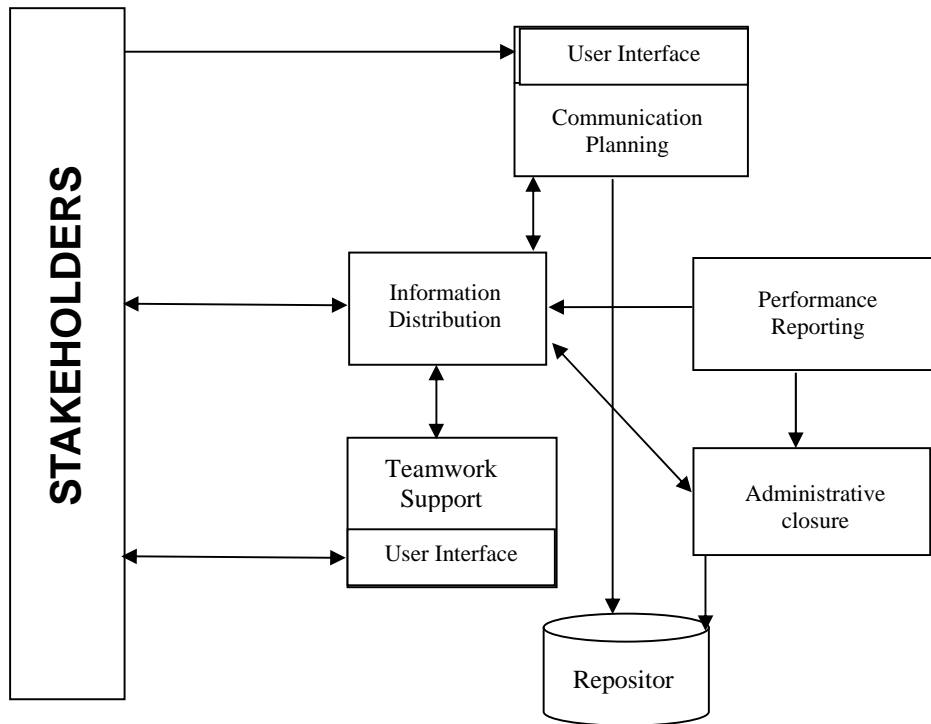


Figure 2: SPM Communication Management function supported by software agent technology.

The interface then assigns a *personal agent* to each person, to be invoked with a user name and password. (For simplicity's sake, the username and password could be the same as a person's network login Id and password, but the choice depends on the individual, or the manager, should he or she decide differently for the sake of security.) Required schedules and resources may be allocated at this stage or omitted if it is assumed that the *Time Management* agent system will do detailed scheduling. *Client* and *task* agents are used for automation where necessary, for example to do resource allocation, or calculations. Once the user has entered the required information into the system, *messaging* agents take the information to a central repository and the *information distribution function*

The *information distribution function* uses an agent team that consists of *messaging* agents, *task* agents, *client* agents and a *team manager* agent. The agent team of this function accepts incoming *messaging* agents from the user interface and uses its own *messaging* agents to interact with the stakeholders, the *teamwork support function* and the *administrative closure function*. It uses *client* agents to gather information from the incoming *messaging* agents and *task* agents to perform information integration and coordination.

In addition to the three primary intelligence properties, *client* (and *task*) agents at this level must also be adaptable in the sense that they remember specific properties of personal agents from previous work on the project, or even from previous projects and as a result, adjust their computing (based on a generic model) to integrate these characteristics. The above intelligence properties also imply flexibility. Developing these agents with the suggested intelligent properties is not a simple task, but since generic patterns exist for many of the other agents, more time can potentially be spent on this part of the development of the SPM tool.

As before, *task* agents are included for specialised computing tasks. For the *information distribution function*, *task* agents may or may not be included at this level, depending on the elaborativeness of the *client agents*. We advocate the use of *task* agents to simplify the design and improve the maintenance of the SPM tool software. As mentioned before, the *client* agent typically has a number of functions including

interacting with (and thus receiving) incoming *messaging* agents, understanding (interpreting) incoming information, translating incoming information to a syntax that makes it processable, processing the incoming information, and deciding on distribution conduct (based on its generic approach to handling information as well as previous knowledge and experience). The *client* agent is also tasked with the responsibility to interact with the outgoing *messaging* agents, which must disseminate the processed information, and must also send the information to the *administrative closure* function that interacts with the central repository as well as with the *teamwork support function*. To simplify the design of a *client* agent, these individual tasks can be designed as *task* agents reporting to the *client* agent via the *team manager* agent.

The *performing reporting* agents, responsible for generating reports, commonly include client agents and messaging agents. Should this function use the same approach as advocated above, *task* agents are also to be included. A decision on whether or not to follow this approach depends on the complexity of the expected reports. The *messaging* agents interact with the *information distribution* function and *administrative closure* functions. In the first instance, mobile agents provide the reports to the *information distribution function* for dissemination, while they also present the report information to the *administrative closure* function for storing and archiving procedures, maintaining system persistence.

The *teamwork support* function is primarily responsible for collaborative scheduling tasks. Agents associated with scheduling are *monitoring* agents, *personal assistant* agents, *client* agents (and *task* agents where applicable), as well as *messaging* agents. *Messaging* agents are defined as before. *Monitoring* agents are responsible for monitoring the incoming messages from *messaging* agents in order to determine the necessity or urgency to suggest new or earlier meeting schedules than those already being scheduled during the previous communication rounds, or by the *teamwork support* function. The primary responsibilities of the *client* and *task* agents are to facilitate teamwork, perform scheduling task on teamwork, and distribute collaborative documents. When an individual team member works on a collaborative document, his or her *personal assistant* agent must be cognisant of any extraordinary circumstances when the user is falling behind schedule. This could for example be done by special-prompting-task-agents asking specific questions or *monitoring* agents comparing set dates to real dates. The *personal assistant* agent passes this information to the (manager) *monitoring* agent, which either sends the agent to the general *client* agent at this level, or makes special suggestions with regard to extraordinary meetings to be scheduled. A user interface is available at this level through which team members can interact with the collaborative task environment.

The *administrative closure* function interacts between the *performance reporting* function and the central repository. It also keeps a history through the use of *monitoring* agents to coordinate incoming reports before storing or archiving the information to the central repository. As expected, this function includes both *messaging* agents and *client* agents (potentially also *task* agents to assist the *client* agents) to coordinate the incoming reports and archiving processes.

4. CONCLUSIONS

The advances in computing technology have evolved over the past decade to a point where distributed computing has become the *de facto* working platform. This has changed the characteristics of SPM, and as a result, the traditional methods and techniques of SPM do not meet the new requirements posed by this new working platform. Software agent technology, although primarily applied to other fields, such as e-commerce, information retrieval and network management, is ideally suited to meeting the new challenges faced by the SPM characteristics such as appropriate tools for effective sharing of information and knowledge among project contributors, as well as efficient distributed task scheduling, tracking and monitoring mechanisms. In this paper we investigated an approach to using software agent technology to address these challenges. We also focussed on one of the key elements of SPM 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 easily be adapted into individual frameworks and then coordinated by an overall multi-agent system to achieve the objectives of SPM. Our framework followed 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 is 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.

5. REFERENCES

- ARIDOR, Y., AND LANGE, D.B. 1998. Agent Design Patterns: Elements of Agent Application Design. In *Proceedings of the 2nd International Conference on Autonomous Agents*. Minneapolis/St. Paul, USA. 108 - 115.
- BALASUBRAMANIAN, S. BRENNAN, R. W. AND NORRIE, D. H. 2001. An Architecture for metamorphic control of holonic manufacturing systems. *Computers in Industry*. Vol 46, 1, 13-31.
- CHANDRASHEKAR, S., MAYFIELD, B. AND SAMADZADEH, M. 1993. Towards automating software project management. *International Journal of Project Management*. Elsevier Science Ltd.
- CHEN, F, NUNAMAKER, J. F., ROMANO, N. C. AND 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. <http://www.alumni.caltech.edu/~croft/research/agent/definition/>
- EARNSHAW, R. AND VINCE, J. (Eds) 2002. Intelligent agents for mobile and virtual media. Springer.
- ELEC 4704 - Software Project Management. Department of Electrical and Information Engineering. University of Sydney. <http://www.ee.usyd.edu.au/elec4704/lec-01.html> Accessed May 2003.
- GARDINER, A. 2002. Implementing PRINCE2 in a Business Change Environment. *Project Magazine*. Vol 4, 2
- GELBARD, R. PLISKIN, N. AND SPIEGLER, I. 2002. Integrating systems analysis and project management tools. *International Journal of Project Management*. Elsevier Science Ltd.
- HALL, G, GUO, Y. AND 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. AND COTTERELL, M. 2002. Software Project Management. Third Edition. 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. AND PATHAK, C.V. 2000. An Application Framework for Intelligent and Mobile Agents. *ACM Computing Surveys*. Vol 32. 1.
- KRUPANSKY J.W. 2003. What is a Software Agent. <http://agivity.com/agdef.htm> Accessed May 2003
- LAZANSKY, J., STEPANKOVE, O. MARIK, V. AND PECHOUCHEK, M. 2000. Application of the multi-agent approach in production planning and modelling. *Engineering Applications of Artificial Intelligence*. Vol 13, 3, 369-376.
- LETHBRIDGE, T. C. AND LAGANIERE, R. 2001. Object-oriented Software Engineering: Practical Software development using UML and Java. McGraw-Hill.
- LINNHOF-POPIEN, C. AND HEGERING, H. 2000. Trends in Distributed Systems: Towards a Universal Service market. Lecture Notes in Computer Science. *Proceedings of 3rd International IFIP/GI Working Conference*, USM 2000 Munich, Germany, Sept 2000.
- 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
- OGHMA: OPEN SOURCE. Types of Software Agents. <http://www.oghma.org/>. Accessed May 2003
- PAI, W.C., WANG, C.C. AND JIANG, D.R. 2000. A software development model based on quality measurement. *Proceedings of the ICISA 13th International Conference*. Computer Applications in Industry and Engineering, 40-43.
- PROJECT MANAGEMENT INSTITUTE (PMI) STANDARDS COMMITTEE. 2000. A Guide to the Project Management body of Knowledge (PMBOK).
- SCHWALBE, K. 2000. Information Technology Project Management. Thompson learning.
- SAUER, J AND APPELRATH, H. 2003. Scheduling the supply chain by teams of agents. *Proceedings of the 36th Hawaii International Conference on System Sciences*. Big Island, Hawaii.
- TAHARA, Y., OHSUGA, A. AND HONIDEN, S. 1999. Agent System Development Method Based on Agent Patterns. In *Proceedings of the 21st International Conference on Software Engineering*. Los Angeles, CA, USA.356 - 367.
- THE STANDISH GROUP. <http://www.standishgroup.com/>
- WOOLDRIDGE M. 2001. An Introduction to MultiAgent Systems. John Wiley AND Sons. Chichester, UK.
- WOOLDRIDGE, M. J., JENNINGS, N. R. 1999. Software engineering with agents: pitfalls and pratfalls. *IEEE Internet Computing*,

20 -27.

B. APPENDIX B

Paper presented at the Global Business and Economic Conference, August 2004, Istanbul, Turkey.

Paper published in the Proceedings of the Global Business and Economic Conference, August 2004, Istanbul, Turkey. *The Business Review*, Cambridge **2**, (1) August 2004.

Software Project Risk Management Supported by Agent Technology

Rita Nienaber, University of South Africa
Elsabe Cloete, University of South Africa
Andries Barnard, University of South Africa

ABSTRACT

The software project management environment is rapidly changing due to globalisation and advances in computing technologies. Currently software projects are developed and deployed in distributed and collaborative environments. This means that traditional project management methods cannot and do not address the added complexities found in a distributed environment, such as effective sharing of information, messages and knowledge among project contributors, as well as efficient task scheduling, tracking and monitoring. Numerous software development projects do not live up to expectations or sadly fail. This is demonstrated by the fact that software projects often do not comply with the traditional standard measurements of success, namely time, cost and specifications. In this paper the utilisation of stationary and mobile software agents is investigated as a potential tool to improve software project risk management. We also propose and discuss a software agent framework to support risk 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 timeously, within budget and to users' satisfaction, while ameliorating the risk associated with software project development.

1 INTRODUCTION

Computing technology is becoming faster, less expensive and more reliable, however, the complexities and risks of software projects continue to increase (Marchewka, 2003.) Over the past two decades, software projects generally failed to satisfy user expectations, were commonly delivered late, and mostly ran over the set budget. The Standish Group (2000) studied 13,522 projects since their famous 1995 report in a follow-up survey, dubbed the EXTREME CHAOS report (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 reported that success rates increased to a third of all projects, but time overruns now measure 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. Initially, techniques utilized in traditional *Project Management* (PM) practices were applied to the development of software projects. However, standard PM methods seemed to lack in the ability to address the unique characteristics of the software development arena (Hughes & Cotterell, 2002). This led to the development of *software project management* as an independent application area and field of study. *Software project management* (SPM) includes amongst other 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.

Practitioners attempted to apply several Software Engineering (SE) principles to different SPM processes in order to address the existing shortcomings in managing software projects (Lethbridge & Laganier, 2000). Standard structured analysis and design methods were explored, and furthermore object-oriented approaches to overcome the aforementioned shortcomings were incorporated (Gelbard et al. 2002; Lethbridge & Laganier, 2000). Standards, such as ISO9001 were formulated, and compliance of the development process to these standards tested. Different software project management approaches based on standards have been developed and are used, namely PRINCE 2 and BS 6079. Yet disappointment remained since many software projects still failed to comply with the triple constraints of scope, time and cost (Oghma: Open Source, 2003). The triple constraints refer to the fact that the failure of software projects can mostly be attributed to the fact that they are not

delivered on *time* and do not meet the expectations of the client (*scope*), and as a result have *cost* implications. 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 traditional focus of SPM processes has shifted from the position afforded to it 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. As organizations continue to invest time and resources in strategically important software projects, managing the risk associated with the project becomes a critical area of concern. While various risk checklists (e.g. the top 10 list of risk factors described by Boehm, 1988) and different frameworks (Marchewka, 2003) have been proposed, there are relatively few tools or methods available to assist project managers to identify, categorize and evaluate risk factors to develop risk management strategies.

Software agent technology offers a promising solution to addressing software 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 team environment, where software agents can support the project manager and team members to monitor and coordinate tasks and in particular to identify, evaluate and monitor risks. SPM skills, especially in the distributed computing environment, are greatly in demand. Moreover, there is a need for technologies and systems to support the 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 to improve management of SPM processes. We specifically concern ourselves with the question of how software agents can be used to improve risk management in a distributed environment. As a result, we propose a software agent framework to support risk management. Although our research is not yet complete, initial indications are that it will be useful in enabling software developers to meet market expectations and to manage risk factors accordingly. This, in turn, will bring about savings in cost, time and effort.

Section 2 of this paper contains a background study of SPM as well as agent computing, and a short discussion regarding each is presented. Section 3 provides a generic-type framework for one of the facilitating SPM processes, namely risk 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 the paper in section 4 by arguing that using agent technology to assist with risk management of SPM, may prove useful to the software developer.

2 BACKGROUND

2.1 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 on the following page illustrates a framework of the key elements in SPM identified by the *Project Management Body of Knowledge (PMBOK)*. We distinguish between three key elements: project stakeholders, project management knowledge areas, and project management tools and techniques. The project *stakeholders* include those people involved in all the different project activities and should, at least, consist of 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 concerned during 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. Stretched across all these knowledge areas are the project management *tool and techniques* (on the right-hand side of the framework diagram). These are used to assist team members and project managers to carry out their respective tasks.

This framework depicts key elements concerned during the management of a single project, in which the SPM manager allocates tasks and gives instructions to various role players. However, traditional tools and techniques supporting the key management areas are no longer adequate in a coordinated, distributed team-development environment (Chen et al., 2003). The 1995 Standish Group study found that the three major factors related to information technology project success were user involvement, executive management support

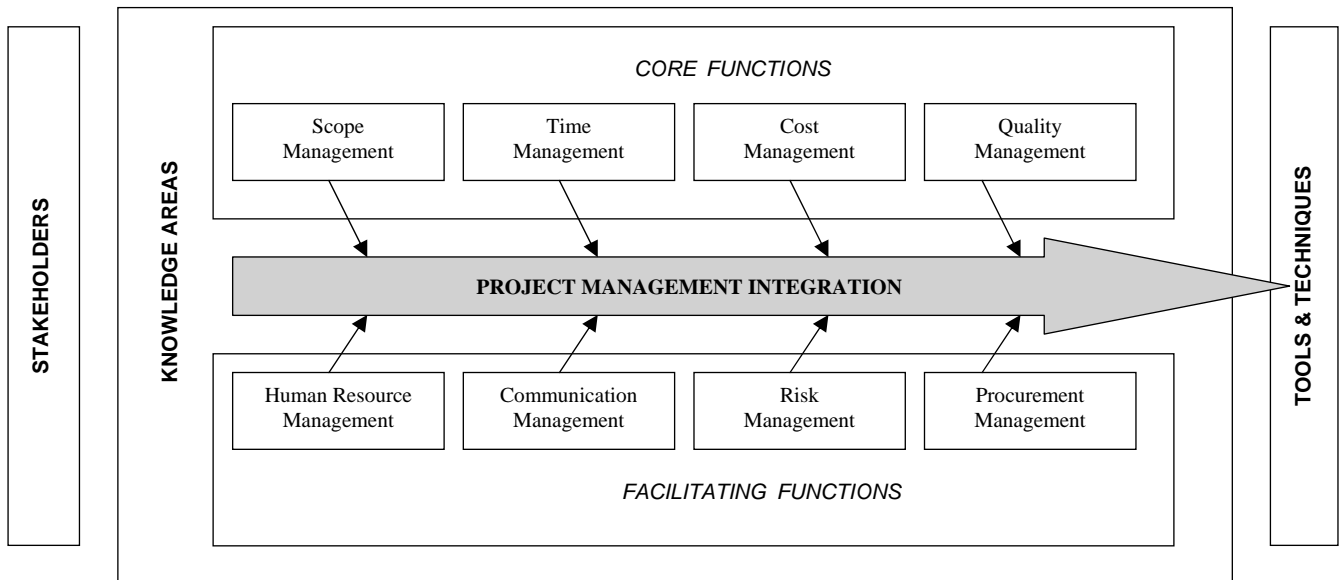


Figure 1: *Software Project Management Framework (adapted from Schwalbe, 2003).*

and a clear statement of requirements (Standish Group, 1995). In the 2000 report executive management support, user involvement, an experienced project manager and a clear statement of requirements top the list of requirements for success (Standish Group, 2000). All these aspects can be addressed and improved by enhancing risk management.

Schwalbe (2003) describes project risk management as *the art and science of identifying, analyzing and responding to risks throughout the life cycle of the project*. The Webster's dictionary defines risk as *hazard, peril or exposure to loss or injury*, whereas the PMBOK defines project risk management as *the systematic process of identifying, analyzing, and responding to project risk*. The objective of risk management is to minimize or avoid the adverse effects of unforeseen events. Many projects do not follow a formal risk management plan, which lead to a state of perpetual crisis. According to Marchewka (2003) the reasons for this include among others that (1) the benefits of risk management is not clearly understood by the project leader; (2) adequate time is not provided for risk management; and (3) not identifying and assessing risk using a standardized approach.

Project management and planning is based on the understanding of various role-players of the current situation, the information available and the assumptions to be made. But as environments may change dynamically, events may not proceed according to plan and various degrees of uncertainties exist which cannot be predicted with total accuracy. To ensure eventual success, those unexpected events must be addressed and managed throughout the life cycle of the project and to ensure that project risk is minimized.

2.2 SOFTWARE AGENTS

A software agent is a computer program that is capable of 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 (Wooldridge, 2001). 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. For example, Krupansky (2003) distinguishes between *ten* different types of software agents, while the Oghma Open Source (2003) web site identifies sixteen different types of software agents. Because 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 may, or may not have, any or a combination of the following characteristics: a user interface, intelligence, adaptivity, flexibility and collaborative properties.

Whether or not an agent has a *user interface*, depends on its potential collaboration 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*, *proactiveness* 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. *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 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 considered to be a prerequisite to identify an agent as being 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).

We consider a stationary agent to be 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. We consider a *mobile agent* to be a software agent that has the ability to transport itself from one host to another in a network. The ability to traverse a network of potential hosts, 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 referred to as a *robot* (Krupansky, 2003).

2.3 AGENT TECHNOLOGY IN SPM

Software agent technology is being explored as a promising way to support and implement complex distributed systems, see Balasubramanian (2001) and Chen (2003). Using software agents in the SPM environment, and in particular in SPM of a distributed environment, is a relative novice field of research and as such, literature in this regard is not commonly available. However, some work on using agents has been performed to address certain aspects pertaining to SPM, refer to Maurer (1996), O'Connor (1999) and Sauer and Appelrath (2003). In this section, the authors briefly consider how agent technology is currently being deployed in SPM by considering some application examples.

The first application that we mention applies risk analysis for a safe, reliable surgical robot system. Korb et al (2003) applies a basic level of risk management in clinical research to implement the robot system RobaCKa for craniotomies. A systematic approach was implemented to support fault-free design, error detection and quality assurance in the design of the robot system. The system was implemented and tested, while further clinical investigations will be carried out in the next two years.

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 and Ritrovato, 2002). Although this project does not relate to risk 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, process monitor and artefact retrieval. Other examples of agent utilization in SPM can be found in Maurer (1996), O'Connor (1999) and Sauer and Appelrath (2003).

3 SOFTWARE AGENT TECHNOLOGY SUPPORT FOR RISK MANAGEMENT

3.1 CONTEXT

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. We argue that agent-based solutions are not only applicable to this environment, but that 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 et al. 2003).

Before discussing our proposed SPM framework, we briefly reconsider the distinct knowledge areas and practices entailed in SPM (illustrated in Figure 1), to emphasise the focus of this paper. The SPM management areas consist of four objective functions and four facilitator functions. The solution presented by Sauer and Appelrath (2003) primarily focuses on the *time management* and certain aspects of the *communication management* functions. Maurer's solution (1996) is applicable to the *scope management*, *time management* and to a certain extent the *communication management* 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 an extensive repertoire of tasks to perform. For the purpose of this paper, we limit our approach to only one of the SPM key processes, namely *risk management*, and describe the agent framework to accomplish the black-box for this process.

Various models or frameworks exist to ameliorate the risk associated with software project development. This basically entails two aspects (Marchewka, 2003), 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. Hughes and Cotterel (2002) identify two major areas including eight distinct functions associated with risk management, based on Boehms model (1988) depicted in Figure 2.

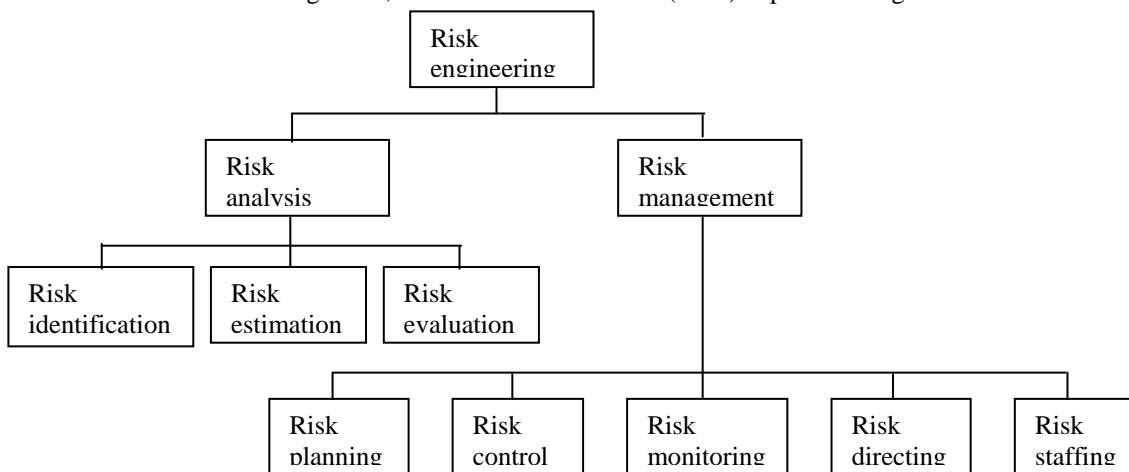


Figure 2: *Software Risk Engineering (Boehm, 1988)*.

For the model we present in this paper, we will adopt a combination of these functions: the *risk identification function* will identify threats, discrepancies and overall inconsistencies with the project plan schedule, budget or time frame, while the *risk estimation and risk evaluation function* will determine what the likelihood is of a particular risk occurring and what impact it will have. It also contains information on how to deal with a particular type of risk, using qualitative and quantitative measures. This will support the project manager to prioritize risks. *Risk planning* alludes to the generation of risk strategies that may be implemented to deal with identified risks and will be stored in the repository. The *risk control, risk monitor, risk directing and risk staffing function* involves implementing those strategies and plans with respect to specific risks and reporting it to the stakeholders.

We propose a risk management model that addresses two categories of risks: first, the risks identified by the stakeholders throughout the project, and second the unexpected risks detected by our agent monitoring system at any time throughout the lifetime of the project. Due to the technical content, as well as the constraints on the length of this report, we will only discuss the management of the first type of risk in this article. The second type of risk will be supported by a risk monitor model based on personal agents that monitor stakeholders as well as teams, and in the event of detecting possible risks, will react accordingly.

To describe how software agents are used to address the different functions of *risk 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 and Lange 1998; Kendall et al. 2000; Tahara et al. 1999) available, accomplish specific agent-associated tasks, such as creation, migration, suspension, and collaboration.

The design of the overall system, based on components (specialised agents) simplify the design and programming of agents. The following specialised working agents are used in our discussion of the risk model that we present in the next subsection. These working agents include:

- *Messaging agent*: 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.
- *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.
- *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*: an agent responsible for monitoring tasks. A monitoring agent is mobile, with intelligence, flexibility and strong collaborative properties.
- *Client agent*: a stationary agent responsible for a specialised task such as information retrieval or gathering. Client agents may or may not have intelligence, depending on their specific task, but must have a collaborative nature to interact with other agents in their agent team.
- *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.

Figure 3 illustrates the main operations in the risk management function and how agent teams cooperate to accomplish the objectives of these operations.

3.2 THE RISK MANAGEMENT MODEL

The software project manager, or other designated stakeholders, interacts with the risk management function through a special user interface. This user interface, which resides on top of the *risk identification function*, uses *personal* agents, *task* agents and *messaging* agents. The interface assigns a *personal* agent to the user who has supervision rights over other *personal* agents. During interaction with the interface, the user defines team members or relevant stakeholders as well as the tasks that are assigned to them, and defines milestones, objectives, risks, et cetera.

The interface then assigns a *personal agent* to each person, to be invoked with a user name and password. (For simplicity's sake, the username and password could be the same as a person's network login Id and password, but the choice depends on the individual, or the manager, should s/he decide differently for the sake of security.) Required schedules and resources may be allocated at this stage or omitted if it is assumed that the *time management* agent system will perform the detailed scheduling. *Client* and *task* agents are used for automation purposes where necessary, for example to effect estimations, risk analysis or calculations.

The *risk identification function* uses an agent team that consists of *messaging* agents, *task* agents, *client* agents and a *team manager* agent. The agent team of this function accepts incoming *messaging*

agents from the user interface and uses its own *messaging* agents to interact with the *stakeholders*, the *risk estimation and risk evaluation function*. It uses *client* agents to gather information from the incoming *messaging* agents and *task* agents to perform *risk identification, risk estimation and risk evaluation*.

In addition to the three primary intelligence properties, *client* (and *task*) agents at this level must also be adaptable in the sense that they remember specific properties of personal agents from previous work on the project, or even from previous projects and as a result, adjust their computing (based on a generic model) to integrate these characteristics. The above intelligence properties also imply flexibility. Developing these agents with the suggested intelligent properties is not a simple task, but since generic patterns exist for many of the other agents, more time can potentially be spent on this part of the development of the SPM tool. *Task* agents are included for specialised computing tasks.

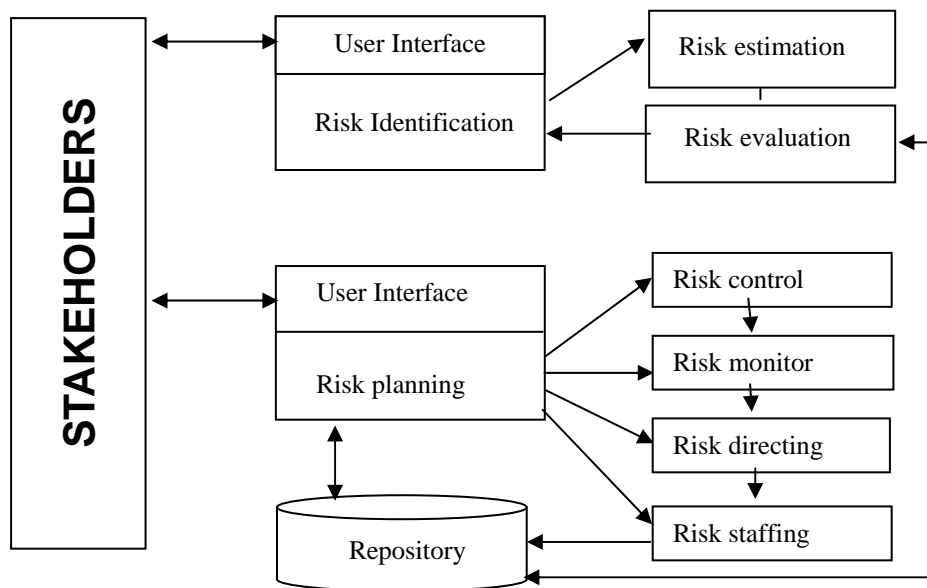


Figure 3: Risk management model for identified risks.

For the *risk planning function*, task agents may or may not be included at this level, depending on the elaborativeness of the *client agents*. We advocate the use of task agents to simplify the design and improve the maintenance of the SPM tool software. As mentioned before, the *client* agent typically has a number of functions including interacting with, /and thus receiving/ incoming *messaging* agents, understanding or interpreting incoming information, translating incoming information to a syntax that makes it processable, processing the incoming information, and deciding on distribution conduct /based on its generic approach to handling information as well as previous knowledge (strategies) and experience/. The *client* agent is also tasked with the responsibility to interact with the outgoing *messaging* agents, which must disseminate the processed information, and must also send and extract when necessary the information to the central repository as well as the *risk control and risk monitor function*. To simplify the design of a *client* agent, these individual tasks can be designed as *task* agents reporting to the *client* agent via the *team manager* agent.

4 CONCLUSIONS

Advances in computing technology have evolved over the past decade to a point where distributed computing has become the *de facto* working platform. This has changed the characteristics of SPM, and as a result, the traditional methods and techniques of SPM do not meet the new requirements posed by this new working platform. Software agent technology, although primarily applied to other fields, such as e-commerce, information retrieval and network management, is ideally suited to meeting the new challenges faced by SPM. Examples of these are appropriate tools for the identification and evaluation of project risk, as well as efficient risk management, tracking and monitoring. In this paper we investigated an approach of using software agent technology to address these challenges. We also focussed on one of the key elements of SPM 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. This research is based upon work supported by the National Research Foundation of South Africa under Grant number 2054319.

5. REFERENCES

- ARIDOR, Y., AND LANGE, D.B. 1998. Agent Design Patterns: Elements of Agent Application Design. In *Proceedings of the 2nd International Conference on Autonomous Agents*. Minneapolis/St. Paul, USA. 108 - 115.
- BOEHM, B. W. 1988. A Spiral Model of Software Development and Enhancement. *Computer* (May) 61-72.
- CHEN, F, NUNAMAKER, J. F., ROMANO, N. C. AND BRIGGS, R. O. 2003. A Collaborative Project Management Architecture. In: *Proceedings of the 36th Hawaii International Conference on System Sciences*. Big Island, Hawaii.
- CROFT, D.W. 1997 Intelligent Software Agents: Definitions and Applications. <http://www.alumni.caltech.edu/~croft/research/agent/definition/>
- ELEC 4704 - Software Project Management. Department of Electrical and Information Engineering. University of Sydney. <http://www.ee.usyd.edu.au/elec4704/lec-01.html> Accessed May 2003.
- GAETA, M. & RITROVATO, P. 2002 Generalized Environment for Process Management in Cooperative Software Engineering. In: 26th Annual International Computer Software and Applications Conference. Oxford England.
- GELBARD, R. PLISKIN, N. AND SPIEGLER, I. 2002. Integrating systems analysis and project management tools. *International Journal of Project Management*. Elsevier Science Ltd.
- HALL, G, GUO, Y. AND 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. AND COTTERELL, M. 2002. Software Project Management. Third Edition. McGraw-Hill. Berkshire, UK.
- IEEE STANDARDS BOARD. 1987. IEEE Standard for Software Project Management Plans. IEEE Std 1058.1-1987. 16pp. ISBN 0-7381-0409-4, SS12138.
- JONES, T. C. 1994. *Assessment and Control of Software Risks*. Upper Saddle River, N.Yourdon Press/Prentice Hall.
- KORB, W. ENGEL, D. BOESECKE, R. EGGERS, G et al. 2003. Risk analysis for a reliable and safe surgical robot system. In: International Congress Series. 1256, 766-770. Elsevier Science B. V.
- KENDALL, E.A., KRISHNA, P.V., SURESH C.B. AND PATHAK, C.V. 2000. An Application Framework for Intelligent and Mobile Agents. *ACM Computing Surveys*. Vol 32. 1.
- KRUPANSKY J.W. 2003. What is a Software Agent. <http://agactivity.com/agdef.htm> Accessed May 2003
- LETHBRIDGE, T. C. AND LAGANIERE, R. 2001. Object-oriented Software Engineering: Practical Software development using UML and Java. McGraw-Hill.
- 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
- MARCHEWKA, J.T. 2003. Information Technology Project Management. Wiley. Danvers, USA.
- O'CONNOR, R. and JENKINS, J. 1999. Using Agents for Distributed Software Project Management. In: *Proceedings of 8th International Workshop on Enabling Technologies*, pp 54-60, IEEE Computer Society Press.
- OGHMA: OPEN SOURCE. Types of Software Agents. <http://www.oghma.org/>. Accessed May 2003
- PAI, W.C., WANG, C.C. AND JIANG, D.R. 2000. A software development model based on quality

measurement. *Proceedings of the ICISA 13th International Conference*. Computer Applications in Industry and Engineering, 40-43.

PROJECT MANAGEMENT INSTITUTE (PMI) STANDARDS COMMITTEE. 2000. A Guide to the Project Management body of Knowledge (PMBOK).

SCHWALBE, K. 2004. Information Technology Project Management. Thompson learning. Course Technology. Canada.

SAUER, J AND APPELRATH, H. 2003. Scheduling the supply chain by teams of agents. *Proceedings of the 36th Hawaii International Conference on System Sciences*. Big Island, Hawaii.

TAHARA, Y., OHSUGA, A. AND HONIDEN, S. 1999. Agent System Development Method Based on Agent Patterns. In *Proceedings of the 21st International Conference on Software Engineering*. Los Angeles, CA, USA.356 - 367.

THE STANDISH GROUP. 1995.CHAOS. <http://www.standishgroup.com/>

THE STANDISH GROUP. 2000. EXTREMECHAOS. <http://www.standishgroup.com/>

WOOLDRIDGE M. 2001. An Introduction to MultiAgent Systems. John Wiley & Sons. Chichester, UK.

C. APPENDIX C

Paper presented at INSITE 2006: The Informing Science and Information Technology Conference, Flagstaff, June 2006.

Paper published in the proceedings of The Informing Science and Information Technology Conference (INSITE 2006), Flagstaff, Arizona, USA. ISSN: 1547-5840, 659-669.

Software Quality Management supported by Software Agent Technology

RC Nienaber, 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 studied 13,522 projects in a survey named EXTREME CHAOS (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 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 *Software 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:

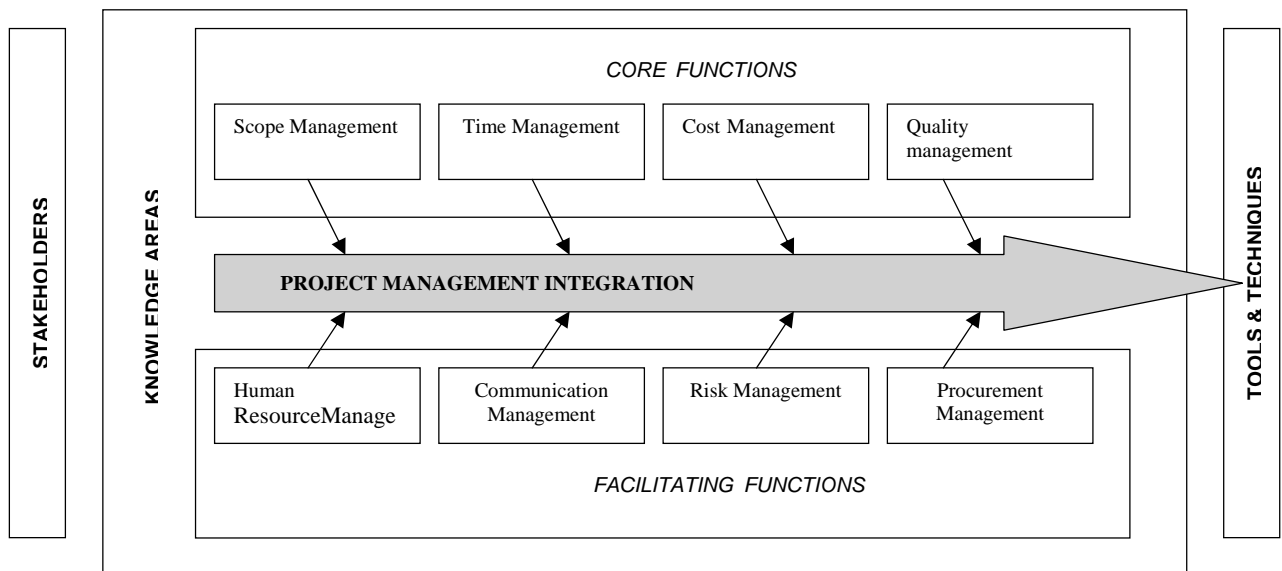
- *Quality planning*: determining which quality standards are relevant to this specific project and deciding how these standards will be met.
- *Quality assurance*: involves evaluating overall performance regularly to ensure conformance to the set standards. Quality audits or reviews can support this 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 et al., 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 & 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 and 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 & 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 et al. 2003).

The first application that we mention, utilizes agents for project planning and process management in a distributed environment. O'Connor & 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 and 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 Appelrath (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 and 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 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.

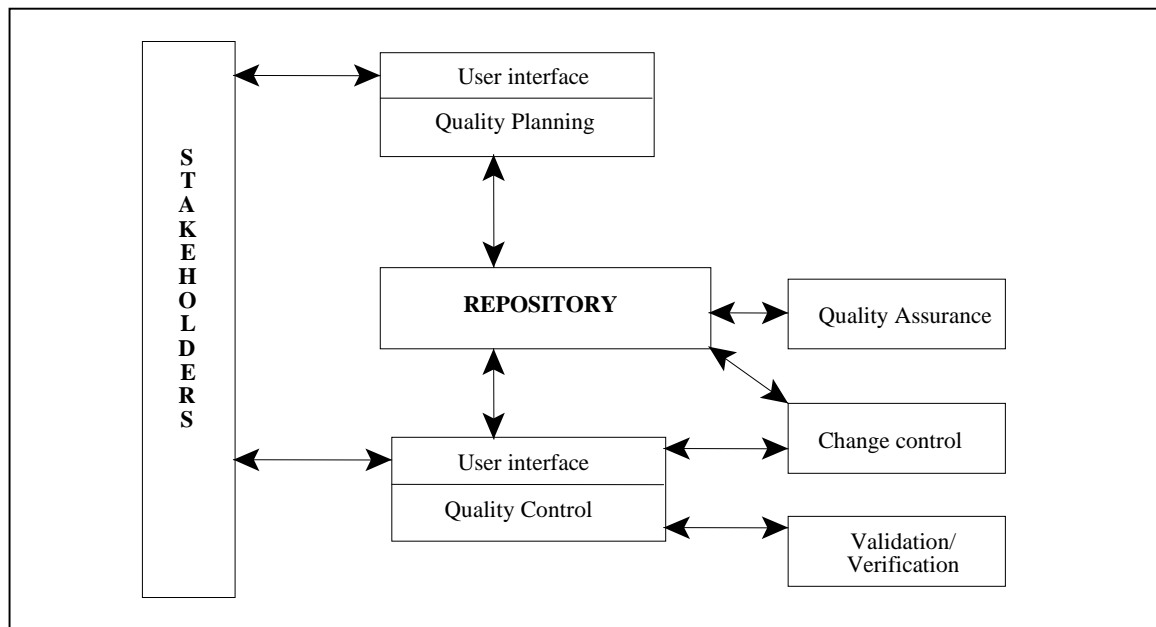


Figure 2: *Software Project Quality Management Framework.*

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. <http://www.alumni.caltech.edu/~croft/research/agent/definition/>
- 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*. Third Edition. McGraw-Hill.
- D'Inverno, M. & Luck, M. (2001) *Understanding Agent Systems*, Springer-Verlag, Berlin.
- 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. University of Sydney. Retrieved May 2004 from <http://www.ee.usyd.edu.au/elec4704/lec-01.html>
- 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*. Vol 32. 1.
- Krupansky, J.W. 2003. *What is a Software Agent*. Website. Retrieved May 2003 from <http://activity.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, *Autonomous Agents and Multi-Agent Systems*, (Sycara, K.(ed)), Kluwer Academic Publishers, 6 (2).
- Pai, W.C., Wang, C.C., & Jiang, D.R. (2000). A software development model based on quality measurement. *Proceedings of the ICISA 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. Vol 32.
- Schwalbe, K. (2004). *Information technology project management*. Third Edition. Thompson Learning.
- Szejko, S. (2002) Requirements driven quality control. *Proceedings of the 26TH International Computer Software and Applications Conference (COMPISA)*
- 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*. John Wiley & Sons. Chichester, UK.



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.

D. APPENDIX D

Paper presented at InSITE 2007: The Informing Science and Information Technology Conference, Ljubjana, Slovenia, June 2007.

This paper was published in the *Interdisciplinary Journal of Information, Knowledge and Management*, Volume 2, 149-162 January 2008.

A Generic Agent Framework to Support 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 timeously, 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-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 (The Standish Group, 1995), 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 Section 2 of this paper, brief information regarding agent technology is provided. Section 3 contains a background study on SPM and a discussion on agents utilised in SPM. In Section 4 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 & Cloete, 2003; Nienaber, Cloete & Barnard, 2004; Nienaber & Barnard, 2005). Finally, Section 5 presents a conclusion.

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 and 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 timeously 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.

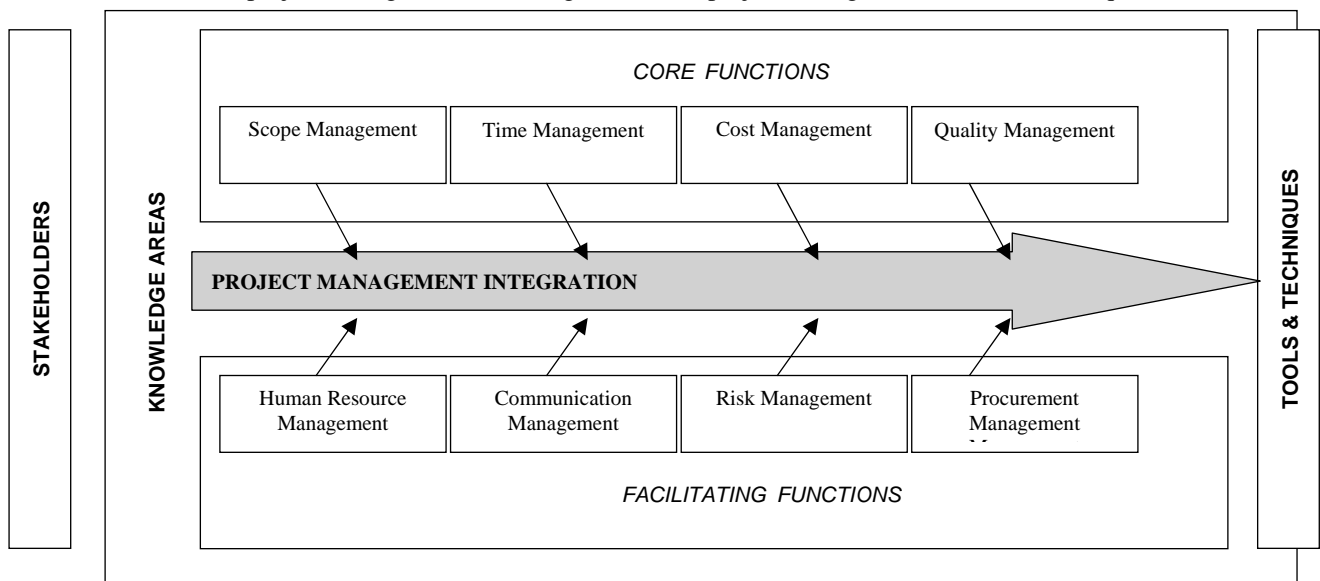


Figure 1: *Software Project Management Framework (adapted from Schwalbe (2006)).*

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 & Appelrath (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 (Burstin, 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, *scope definition* involves the division of major project deliverables into smaller and more manageable components and *scope 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 *quality 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 7 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 (2006) 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. The following table 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).

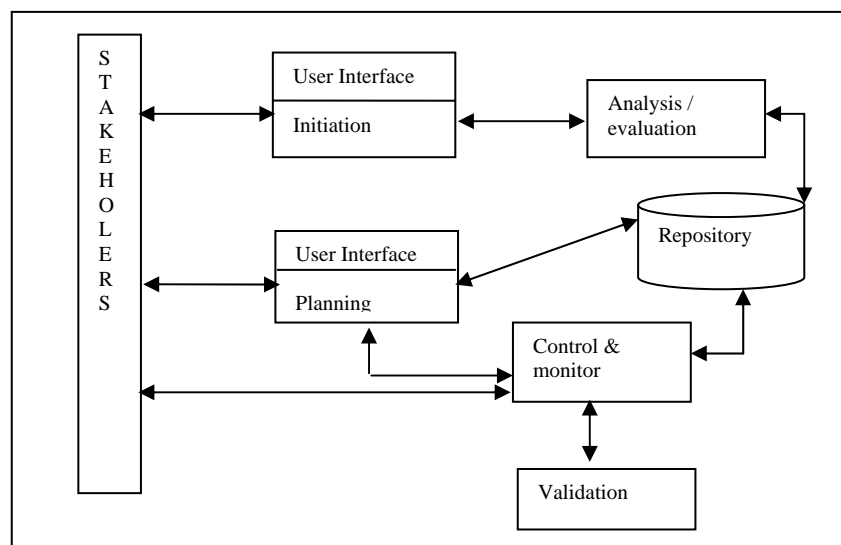


Figure 2: Generic model for SPM processes

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.

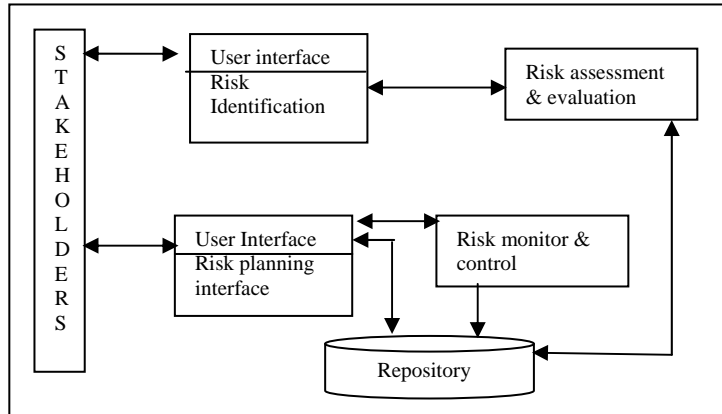


Figure 3. Software risk management.

The generic model as depicted in Figure 2 was instantiated to one key area, namely risk management, resulting in Figure 3 above. 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 are shown on the following page in Figure 4.

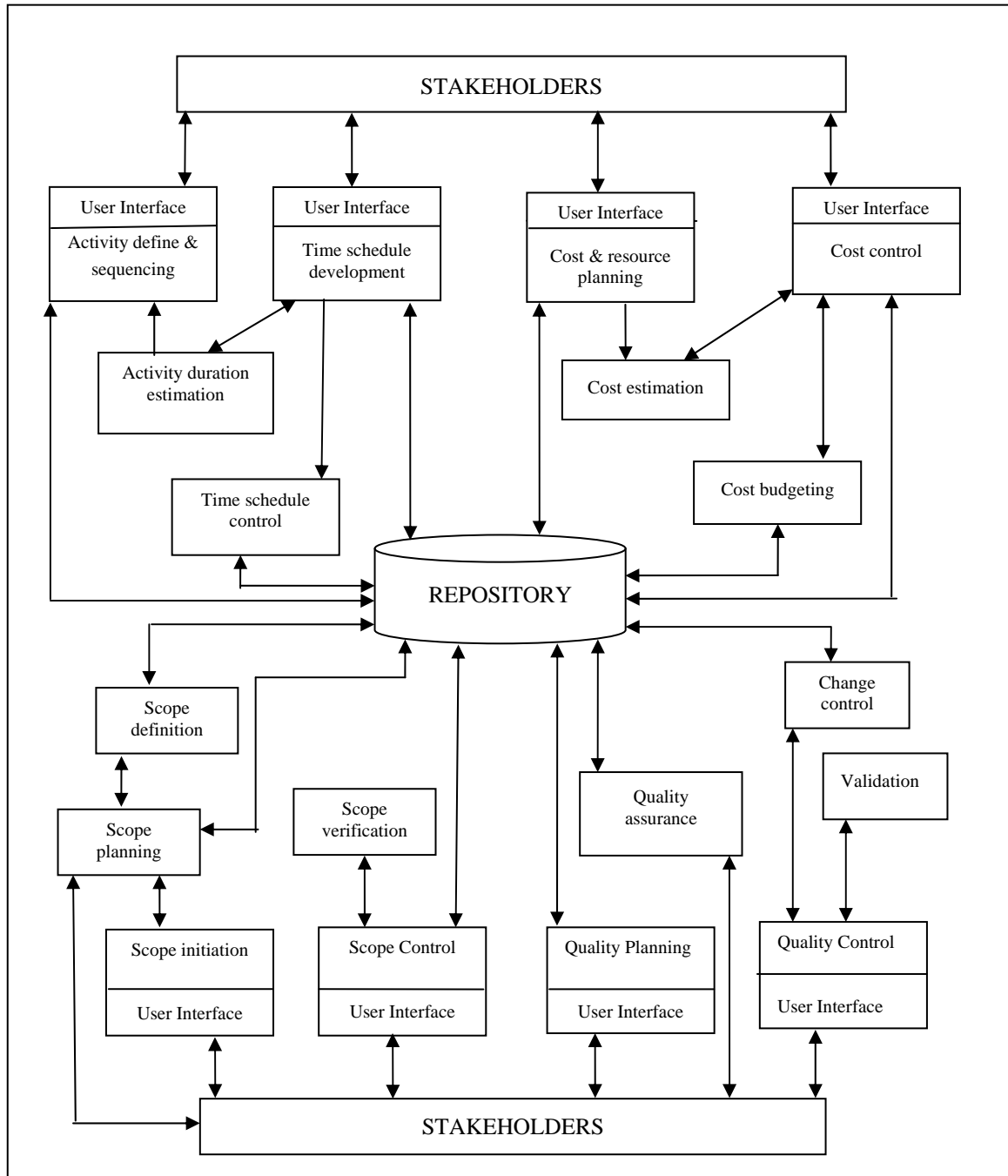


Figure 4: Conceptual model for the core functions: time management, cost management, quality management & scope management.

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

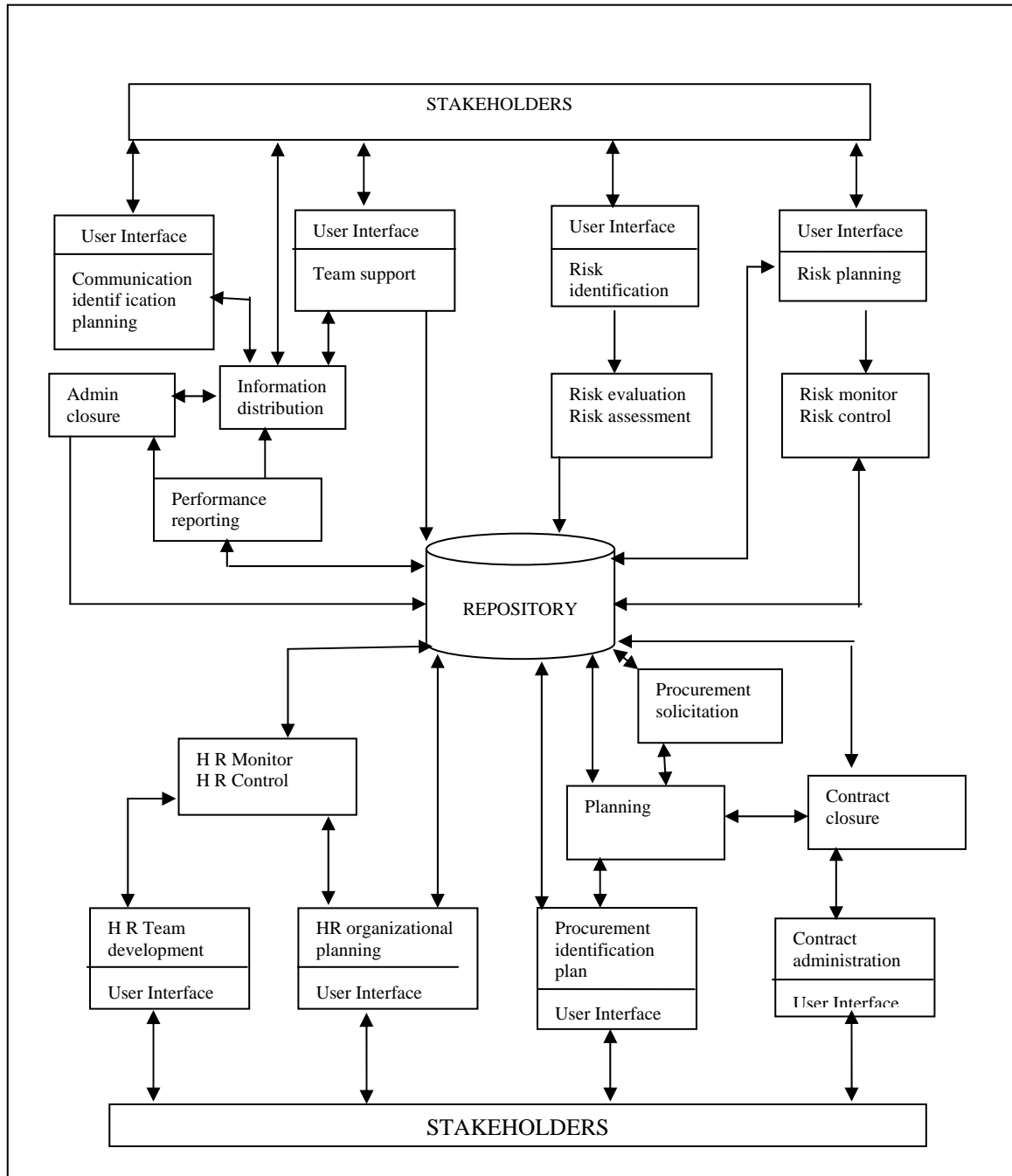


Figure 5: Conceptual model for facilitating functions: communication management, risk management, procurement management and human resource management

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, 2002), it is viable to implement the system in Java. JADE (Java Agent Development Framework) is a software framework to de-

velop 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 (ISO17799, 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 are 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.

Table 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). A Spiral Model of Software Development and Enhancement. *Computer* (May) 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 Vol 8 issue 3 pp 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. and Luck, M. (2001), *Understanding Agent Systems*, Springer-Verlag, Berlin.
- 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. 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*. Fourth Edition. 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. 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*. Vol 32. 1.
- Krupansky, J. W. (2003). What is a Software Agent. Retrieved October 12, 2005 from <http://activity.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 & 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*, pp 452-459
- Nienaber, R C & Barnard, A. (2005). Software Quality Management supported by Software Agent Technology. *Issues in Informing Science and Information Technology*, pp 659-670
- 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*. Fourth Edition. 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*. Second Edition. McGraw-Hill.
- Wooldridge, M. (2002). *An introduction to multi-agent systems*. John Wiley & Sons. Chichester, UK.

E. APPENDIX E

Paper presented at IADIS, 2008: The International Applied Computing Conference,
10 -13 April, Algarve, Portugal.

This paper was published in the conference proceedings. ISBN 978-972-8924-52-2

SOFTWARE AGENT TECHNOLOGY SUPPORTING RISK MANAGEMENT IN SPM

Nienaber RC, Smith E, & Barnard A,
University of South Africa
South Africa

Van Zyl T
The Meraka Institute
South Africa

ABSTRACT

Globalisation and advances in computing technologies have 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. Thus, there is a need for new methods and measures to support the software project management process.

In this paper software agents are investigated as a potential tool to assist with the enhancement of software project management processes. In particular the authors propose to enhance software project management processes with a software agent framework, which forms part of the SPMSA model. A prototype is developed for the risk management function area of this model, to prove that the model, and specifically the agent framework, is not merely a theoretical concept, but can indeed be implemented. This research shows promise of significant results 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, Risk management

1. INTRODUCTION

Software Project Management (SPM) has become a critical task in many organizations. Over the past years, development of software projects regularly failed to come up to user expectations, were commonly delivered late, and mostly ran over the set budget. Much of this still holds true in the present context, and these issues have to be addressed in concrete terms (Chen, Nunamaker, Romano & Briggs, 2003). As a result the field of software project management (SPM) is receiving increasing attention and various methods and techniques are utilized to optimise the SPM processes.

SPM involves the management of all aspects and issues that are involved in the development of a software project, namely identification of scope and objectives, 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 (Hughes & Cotterel, 2006). Furthermore, SPM processes comprise their own unique features. Characteristics unique to software projects are invisibility, complexity, conformity and flexibility. These aspects contribute to the difficulty in pinpointing a software project as an exact task with a specific beginning, an end and deliverables. The unique nature of SPM is seen as a contributing factor to the difficulties experienced with managing software projects and the failure of such projects.

During the past decade computer technology expanded and management and control functions were automated and supported by software tools and techniques in an effort to support SPM management and control (Chen et al., 2003). Currently the SPM environment is still changing due to business globalization and information technology advances that support distributed and virtual teams and projects (Chen et al., 2003). The increasing number of distributed projects involving software project collaborators from different locations, organizations and cultures, changes the SPM paradigm of a traditional project focussing on a single project executing at a specific location (Jonsson, et al., 2001; Smits & Pshigoda, 2007). The focus of SPM processes has clearly shifted from the position that it held two decades ago. Consequently, tools for effective sharing of information and knowledge among

project contributors, as well as efficient task scheduling, tracking and monitoring are needed. High levels of collaboration, task interdependence and distribution have become essential across time, space and technology (Chen et al., 2003).

The need for flexible management of ever changing organizational structures such as dealt with in SPM, are suitably addressed by the computational mechanism of agent systems (Jennings, 2001). Jennings and Wooldridge (1998) define a software agent as an autonomous system, capable of flexible autonomous action in order to meet its design objectives. Software agents are appropriate in highly *dynamic, complex, centralised as well as distributed situations* (Butte, 2002). 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, et al., 2003). Agent computing provides explicit benefits for *open and dynamic environments*. In particular software agent technology provides a useful paradigm for the use of *distributed computational resources*. Mobile agents enable a shift in the communication paradigm of distributed systems from data shipping to function shipping (Butte, 2002). 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 will enable the designer to attain a higher level of abstraction. This *autonomy* reduces network load and communication overhead in distributed applications. Furthermore agents are naturally *heterogeneous*, thus mobile agents can execute on different hardware and software platforms. Software agent technology therefore 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, to validate and verify, as well as change control.

In previous work we reported on the development of a model for SPM, where the SPM processes are supported by software agents (Nienaber & Barnard, 2007). The model entitled “SPMSA” (Software Project Management supported by Software Agents) aims to enhance the SPM processes by addressing the intrinsic unique aspects of SPM. This model is unique in that it aims to support and enhance the *entire* environment of the SPM arena. The purpose of this paper is to report on work-in-progress regarding implementing one section of the SPMSA model i.e. the risk management function. This implementation serves to illustrate that the proposed SPMSA model (see reference for more detail on this model) can be implemented and is not merely a theoretical concept. Due to paper length it is not possible to discuss the implementation of the *entire model*.

2. OVERVIEW OF THE SPMSA MODEL

The main goal of the proposed SPMSA model is to support the teams and individual team members in the SPM environment while executing their tasks. The team leader, teams and individual team members will be supported during each process of software project management to simplify the environment, eliminate the complexities, enhance coordination and communication, implement dynamic changes in the system and support task scheduling. Figure 1 on the following page presents a conceptual view of the proposed model.

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. These are used to assist team members and project managers in carrying out the core and facilitating functions. These form the basic key function areas of SPM.

Each key function has a number of basic phases. On close inspection overlapping phases can be identified, as executed in each of these processes. An abstraction of these functions may be mapped to a generic model, containing overlapping phases for each function or process of SPM (Nienaber & Barnard, 2007 for more detail).

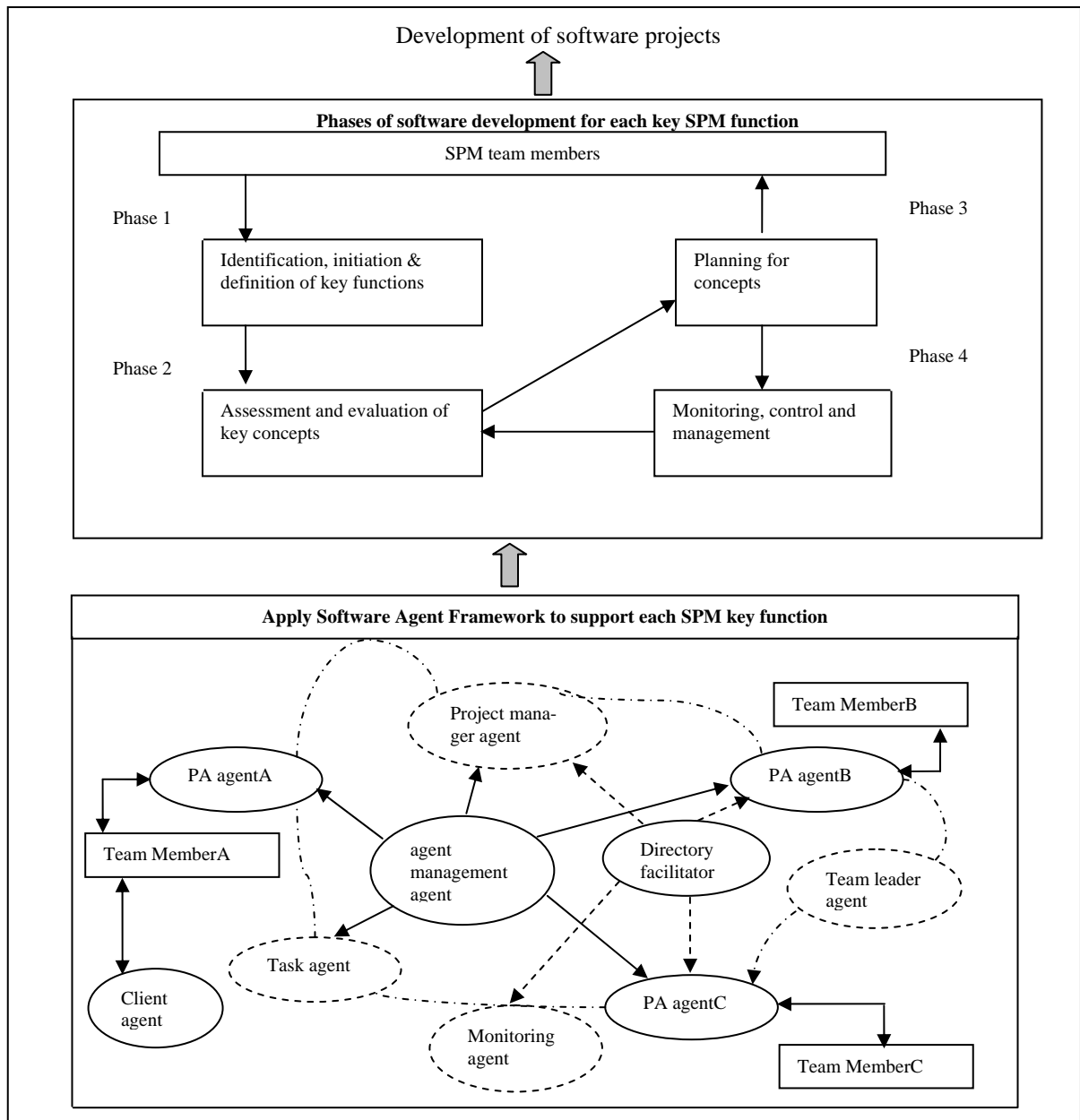


Figure 1. Conceptual view of the SPMSA model

A generic model of software development for each key function is represented by the *top part* of figure 1 i.e. the different phases. During phase 1 (Identification, initiation and definition of key functions) elements concerning each key function are defined, scrutinized and initiated. In phase 2 concepts of the key function concerned is assessed and evaluated. The planning for the “concepts of the particular process” of the key function is conducted in phase 3. Finally, in phase 4 the different functions associated with each function are monitored, controlled and managed.

Although a few additional functions may exist according to the specific key function, all functions have these basic phases in common. During the implementation of these phases the phase will be tailored to each individual key function, i.e. cost or time management. The upper section of figure 1 therefore represents the SPM processes that will be supported by an agent framework.

Each of the key functions of SPM will be supported by a combination of one or more of the agents as shown in figure 1 – *bottom part*. The software agents will support the generic functions for each of the key SPM processes (with minor practical differences, i.e. risk or time initiation). The various phases of SPM will be supported by agents or teams of agents (the agent framework). For illustrative purposes this basic configuration is represented as a conceptual view of the operational environment of 3 team members, which will probably be geographically dispersed, and is depicted in figure 1.

To describe how software agents can generically be used to address different functions of SPM, we use a set of *agent teams* to address the functions and then define specialised software agents operating

within these teams, or on their own where applicable. For a detailed discussion on software agent technology, the interested reader is referred to Wooldridge (2002) and Butte (2002). The following specialised working mobile and stationary agents are used:

A *Personal Assistant Agent (PA agent)* supports each individual team member to accomplish his or her tasks by providing maximum assistance, as well as providing an interface between the team member and other agents. The PA agent is not computer-bound, but human-bound, as its stakeholders may be required to work on different computers when working in a distributed environment.

The *Task Agent (TA)* is an agent that supports a specific project task. This agent collaborates with other objective and facilitator functions to support a specific task. This mobile agent is commonly invoked by a PA agent to allow a stakeholder to work on a specific task.

The *Client Agent (CA)* is a stationary agent responsible for a specialized task, such as information retrieval.

The *Monitoring Agent (MA)* is a mobile agent responsible for monitoring tasks, reporting back to enable scheduling, rescheduling of tasks, as well as the notification of stakeholders.

The *Directory Facilitator (DF)* provides a yellow page functionality that assists agents to discover services provided by one another.

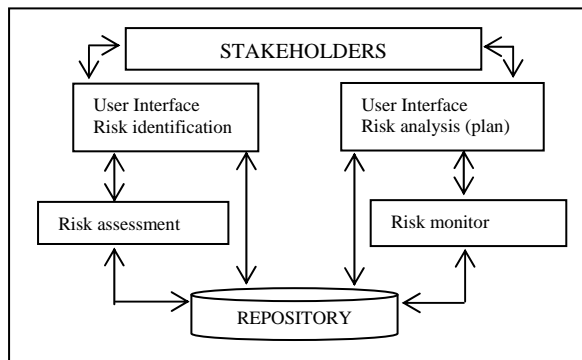
The *Agent Management Agent (AM)* is responsible for managing a team of agents, ensuring coordination between the sub-tasks of the different members of a team to accomplish the objective of the agent team. This agent enables communication, mobility, instantiation and destruction amongst other tasks.

Team Leader (TL) is an agent that is responsible for managing a team of agents, ensuring coordination between the sub-tasks of the different members of a team.

Project Manager (PM) is an agent that takes on the project manager role, assists with the creation of the project, initial specification of the tasks, and allocation of tasks to personnel.

3. SOFTWARE PROJECT RISK MANAGEMENT

This paper focuses on the *risk management* function area (part of the facilitating functions – as mentioned in the previous section) of the proposed model only. For the proposed of this study, risk analysis comprises the *risk identification function* (i.e. identifies potential risks), the *risk assessment function* (i.e. determine what the likelihood of a particular risk occurring is), the *risk planning and analysis* (i.e. determine risk strategies to deal with identified risks), and the *risk monitor function* (i.e.



monitor the environment to pinpoint possible changes that needs attention). Figure 2 depicts a graphical representation of the different risk management stages as described above. Each of these key 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. This will be further addressed by the prototype in the next section.

Figure 2. Risk management stages

4. PROTOTYPING THE RISK MANAGEMENT FUNCTION AREA

The purpose of this section is to illustrate how one functional area of the SPMSA model, namely risk management, can be implemented. Other functional areas of the model will be implemented in a similar way.

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, 2002), it is viable to implement the risk management function area of the proposed model 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 JADE platform allows for easier communication by adhering to the FIPA standard for Agent Communication referred to as FIPA-ACL. Agent interaction is explicitly handled by the JADE framework through the Directory Facilitator (DF). The DF acts as a yellow page directory to enable the discovery of agents and agent services. The entire process of agent management including agent mobility, suspension, awaking, creation and destruction is handled by the Agent Management System (AMS). The JADE container provides a context for agent existence and can be extended by other containers to form a distributed environment (Bellifemine et. al., 2001). The risk management stages as illustrated in figure 2 will be supported by a team of agents on the JADE platform.

The *risk identification phase* entails that specific risks are identified by the project manager and the team members. The *Project Manager agent* uses the *Directory Facilitator agent* to identify *Personal Assistant agents* of all the members of the team. The project manager then *identifies possible risks* for this project. The *Agent Manager agent* is responsible for managing the team of agents, ensuring coordination between the sub-tasks, communication between agents and location distribution of agents. The project leader defines the parameters of the project with the assistance of the *PA agent*, adds tasks and subtasks, and allocates tasks to team members, with the support of the *PA agent*. The monitoring agent will be responsible for monitoring tasks, reporting back to the *PA's* where rescheduling of tasks as well as the notification of stakeholders can take place. Task documents, attached to a specific task,

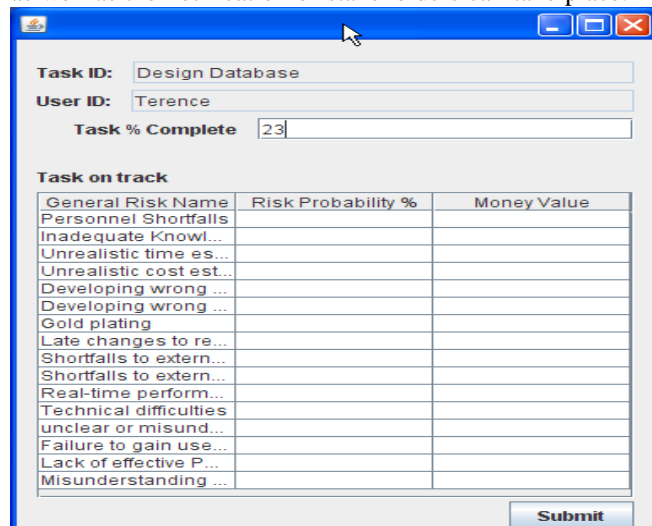


Figure 3. Task status

will also be monitored. The task agent will traverse the distributed network for input of all team members on the selected risks, which will enhance the process by sending the functionality to the various team members. Network load is lightened and communication overhead lessened. Task documents will also automatically be distributed over the distributed network, lessening the work load of each team member.

During risk assessment the task agent further traverse the distributed network of team members, communicating with each team members' personal assistant agent. Each team member will allocate a weight to relevant risks in the form of the probability of occurrence on a scale of 1-100. The task agent will perform the calculation of risk probability for each risk identified, as well as correlating the input of all team members. The two columns in figure 3, Risk probability and Money value, will be populated once it is calculated by the task agent. The team members will continuously be informed of the risk probability and monetary value implications if known at this stage. Furthermore, each team member will be prompted on the percentage that each task is completed, at regular time intervals. The task documents will also be attached to a task. Thus this function will enhance the risk management function by continuously updating team members on the probability of a certain risk occurring. The status of each task will also be monitored by the task and monitoring agent. In other words if a task runs late, the preceding tasks will be sent a message that there is a problem, and when the task is finished completed deliverables and documents are automatically distributed amongst the team.

The *risk monitor and control phase* will provide the team and the team leader with information on the status of each specific task; a warning message if tasks or deliverables are overdue or on schedule; the probability of occurrence of identified risks, on a scale of 1 – 100. Due to paper length only one screen of the prototype is shown, namely figure 3, which report on the task status. Task Design Database is on track and thus no action needs to be taken. The SPM process is enhanced by the additional information supplied to the team and team leader on a daily basis. The team leader will be

informed of the status of all tasks of the specific project. If a task falls behind schedule immediate action can be taken. Communication overhead and network load are lessened. Distributed teams can also communicate and coordinate through this heterogeneous nature of the agent system. Note that the risk planning phase, does not yet form part of the scope of the prototype. This prototype is being refined and will be evaluated in a real-life scenario, as well as verified by comparing it to an ISO standard.

5. CONCLUSION

In this paper we illustrated that our SPMSA model can be implemented by agents and we highlighted the benefits of such a supporting agent framework in the SPM arena. Our prototype currently implements only certain sections of the model as it is implemented as 'proof of concept' but it can be expanded to include the entire model as well as all areas of SPM.

Our research is aimed at software practitioners and software developers, but will also be beneficial to researchers working in the field of SPM. The development of software projects supporting crucial business activities may be utilized to attain a competitive advantage for that organization. Thus, the quality of the software development process, as well as improvements in the development of project management software may potentially result in significant improvement in software quality (Schwalbe, 2006).

REFERENCES

- Bellifemine, F., Poggi, A. and Rimassa, G. 2001. JADE: a FIPA2000 compliant agent development environment, *Proceedings of the fifth international conference on Autonomous agents*, pp 216-217.
- Butte, T. 2002. Technologies for the development of agent-based distributed applications. *Crossroads*, Vol 8. No 3, pp 8-15.
- Chen, F, Nunamaker, J., Romano, N. and Briggs, R. 2003. A Collaborative Project Management Architecture, *Proceedings of 36th Hawaii International Conference on System Sciences*. Big Island, Hawaii, pp .
- Hall, G., Guo, Y. and 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, pp.
- Hughes, B. & Cotterel, M. 2006. *Software project management*. Fourth Edition, McGraw-Hill, London.
- Jennings, N.R. 1999. Agent-Based Computing: Promise and Perils. *Proceedings of 16th joint Conference on Artificial Intelligence (IJCAI-99)*, pp 1429-1459.
- Jonsson, N., Novosel. D., Lillieskold & Eriksson, 2001. Successful Management in Complex Multinational R & D Projects. *Proceedings of the 34th Hawaii International Conference on System Sciences*. Maui, Hawaii.
- Nienaber, R. C. and Barnard, A. 2007. A Generic Agent Technology Framework to support the Various Software Project Management Processes. *Interdisciplinary Journal of Information, Knowledge and Management*, Vol 2, pp 149-162.
- Schwalbe, K. 2006. *Information technology project management*. Fourth Edition, Thompson Learning, Canada.
- Smits, H. and Pshigoda, G. 2007. Implementing Scrum in distributed software development organization. *In Proceedings of AGILE2007 Conference*, pp 371-375, Washington, DC.
- Wooldridge, M. 2002. *An introduction to multi-agent systems*. John Wiley & Sons, Chichester, UK.

F. APPENDIX F

Paper was submitted to the International Journal of Information Management in June 2008.

Enhancing and supporting SPM: the Software Project Management Supported by Software Agents model

¹RC Nienaber & ²E Smith

School of Computing, University of South Africa, Pretoria

¹nienarc@unisa.ac.za, ²smithe@unisa.ac.za

Abstract

The scope, environment and implementation of traditional software projects have changed because of various factors such as globalisation, advances in computing technologies and, last but not least, the development and deployment of software projects in distributed, collaborative and virtual environments. As a result, traditional project management methods fail to address the added complexities found in this ever-changing environment.

In this paper the authors propose a software project management (SPM) model, entitled SPMSA (Software Project Management Supported by Software Agents), that aims to enhance SPM by taking the unique nature and changing environment of software projects into account. The SPMSA model is unique in that it supports the entire spectrum of SPM functionality, thereby supporting and enhancing each key function with a team of software agents. The project manager and the individual team members will be supported during software project management processes to simplify their tasks, eliminate the complexities, automate actions and enhance coordination and communication. At the same time, virtual teamwork, knowledge management, automated workflow management and process and task coordination will be supported.

The phases of the SPMSA model also compare favourably with the basic phases of software development as prescribed by the ISO 10006:2003 standard for projects. It can therefore be concluded that the SPMSA makes a fresh contribution to enhancing SPM by utilising software agent technology.

Key terms

Software projects; software project management; software agent technology

1. Introduction

Most business undertakings these days are commonly supported by software applications. The quality, effectiveness and efficiency of these applications determine the success or failure of many business solutions. As a result, businesses often find that they need to attain a competitive advantage through the development of software projects that support crucial business activities. The quality of the software development process plays a key role in the quality of the software implementation. Improvements in the development of project management software used to manage software development can result in significant improvement in software quality (Schwalbe, 2006).

The literature reveals that ongoing research aims to address the existing shortcomings in managing software projects (Roy, 2004; The Standish Group, 2005). Practitioners have attempted to apply several software engineering principles to different software project management (SPM) processes (Lethbridge & Laganier, 2001). They have explored standard structured analysis and design methods and also incorporated object-oriented approaches to overcome the aforementioned shortcomings (Gelbard, Pliskin & Spiegler, 2002; Hughes & Cotterell, 2006). Different standard project management approaches exist, which are applicable to different areas of software project management, such as PRINCE 2 and BS 6079:1996 (Hughes & Cotterell, 2006). Yet many software projects have 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 the fact that they are not delivered on *time* and do not meet the expectations of the client (*scope*), with the result that they have *cost* implications.

The abovementioned problems can be ascribed to various factors, but mainly to the fact that the SPM environment has changed over the past decade, and is still rapidly changing due to globalisation and advances in computing technology (Zanoni & Audi, 2003). The traditional single project, which was commonly executed at a single location, has evolved into distributed, collaborative projects deployed in distributed and collaborative environments. This means that traditional project management methods cannot address the added complexities found in a distributed environment, such as efficient task scheduling, tracking and monitoring, as well as effective sharing of information and knowledge among project contributors. Therefore there is a clamant need for managing software project risks in such a way that this complex distributed environment is addressed and supported optimally.

The purpose of this paper is to propose an SPM model – entitled Software Project Management Supported by Software Agents (SPMSA) – that enhances SPM processes by incorporating a software agent technology framework (Chen, Nunamaker, Romano & Briggs, 2003). The first part of this paper highlights the unique features of the project management environment, as well as the changing nature thereof. In the second part of the paper, the suitability of software agents to support SPM is explained, and the third part is devoted to a discussion of the SPMSA model. The paper culminates in a verification of the SPMSA model against the ISO standard 10006:2003.

2. Unique nature and changing environment of SPM

Software project management (SPM) differs from general project management as certain inherent characteristics are unique to software development (Brooks, as quoted by Hughes & Cotterell, 2006). These characteristics are invisibility, complexity, conformity and flexibility. *Invisibility* implies that the development process of the software cannot be seen visually, which makes controlling, monitoring, measuring and estimating project progress difficult. Furthermore, *complexity* of software projects is increased in that software projects include not only the development of a system, but also the implementation and maintenance of a system that may be distributed and which interfaces with many existing systems. The *conformity* of software is essential because software projects involve a variety of resources where the software is expected to conform to the requirements of humans and organisations. Finally, *flexibility* is needed as software systems are required to conform to the standards of the organisation and are therefore subject to a high degree of change. These aspects contribute to the difficulty in clearly pinpointing a software project as an exact task with a specific beginning, an end and deliverables.

The dynamic environment of SPM adds to the complexity of these systems, resulting in higher levels of interconnectivity, higher levels of data and knowledge sharing, task tracking and monitoring. These issues should be supported optimally by SPM processes to enable project managers to concentrate on crucial issues, thus striving for less failure and higher success rates in software projects.

3. Software agent technology

SPM practices should take full cognisance of the unique nature and changing environment of SPM. Traditional SPM methods do not address the added complexities found in an ever-evolving distributed environment. The authors investigated the possibility of using software agent technology to address SPM problems in a distributed environment to enhance SPM processes.

Table 1 lists the unique requirements of SPM and how agent technology may address these.

Table 1 SPM features to be addressed by agent technology

SPM	Software agent technology
Changing environment of SPM systems leads to a <i>complex distributed environment</i> .	Agents allow distribution and communication over a geographical area, irrespective of the geographical location. Agents overcome network latency by executing locally and reduce network load (Lange & Oshima, 1999). Parallel execution enables tasks to be executed in parallel at different workstations.
SPM distributed environments may require and incorporate mobile devices and <i>fragile network connections</i> .	Agent systems can incorporate large network systems and mobile devices. Tasks can be embedded into mobile agents, which can traverse the network and execute asynchronously and autonomously, without relying on a continued connection (Lange & Oshima, 1999).
A distributed environment requires a high level of <i>collaboration and cooperative problem solving</i> between teams and team members.	Teams of agents can coordinate actions toward a similar goal in a distributed environment. Software agent technology provides a natural metaphor for support in a team environment, where software agents can traverse the network in order to monitor and coordinate events (Wooldridge, 2002). Communication and cooperation is strongly supported by agent teams.
A distributed environment results in virtual, <i>dynamically changing</i> collaborative teams.	Agents adapt dynamically to changes. Agents are aware of their environment and can respond to changes in it. Agents have the computational mechanisms for flexibly forming, maintaining and disbanding organisational structures (Jennings, 2001)
Collaboration between team leader and distributed team members requires continuous <i>control, monitoring and measurement</i> (invisibility aspect).	Agents are perfectly suited to control, monitor and measure elements in a distributed environment (Braun et al., 2001).
A distributed environment requires <i>heterogeneous technology</i> and databases that have to interact and share information.	Agents are naturally heterogeneous and mobile and can execute on different hardware and software platforms (Lange & Oshima, 1999).
The changing SPM environment requires <i>flexibility and conformity</i> of the system.	Agents adapt dynamically to changes in their environment, therefore this aspect will be excellently supported (Kotz & Gray, 1999).
Virtual software project teams over dispersed environments need to <i>access information and documents</i> .	Agents support the distributed retrieval and dissemination of information and documents and can automate routine tasks (Maes, 1996; Green et al., 1997).

From table 1 we can conclude that software agent technology provides a suitable framework for supporting and possibly enhancing SPM processes in a complex distributed environment. The need for flexible management of ever-changing organisational structures, such as are dealt with in SPM, is suitably addressed by the computational mechanism of agent systems (Jennings, 2001). As stated in table 1, agent behaviour can be used to support the individual team members in numerous tasks, such as coordination and cooperation with team members, document retrieval and distribution, workflow monitoring and control, scheduling and organising meetings, and reminders for tasks and overdue dates or deliverables.

4. The SPMSA model

Having established that agent technology is suitable for supporting the SPM environment, the authors compiled a comprehensive model of SPM functionality to be supported by software agent technology. The

SPMSA model enhances and supports *all* core and facilitating functions of SPM by utilising an agent framework to enhance the SPM processes. This model thus addresses the *entire* spectrum of software project management.

4.1 Conceptual view of the SPMSA model

Having investigated and delineated all SPM processes, the authors defined the SPM core and facilitating functions, as depicted in table 2; these form the basis of the proposed SPMSA model (Schwalbe, 2006; Hughes & Cotterell, 2006; Boehm, 1991; Olson, 2004 and Marchewka, 2003).

Table 2 SPM core and facilitating functions

Scope management	Time management	Cost management	Quality management	HR management	Communication management	Risk management	Procurement management
Initiation	Activity definition				Identification & planning	Risk identification	Procurement planning
Planning	Activity sequencing ; duration estimation	Resource planning	Planning	Organisational planning	Team support	Risk analysis & prioritisation	Solicitation planning
Definition	Time schedule development	Cost estimation	Assurance	Team development & staff acquisition	Information distribution	Risk management planning	Solicitation & source selection
Verification	Time schedule control	Cost budgeting	Control	Management: monitoring & control	Performance reporting	Monitoring	Contract administration
Change control		Monitoring & control			Admin closure	Resolution	Contract closure

When examining table 2 closely, *overlapping phases* can be identified, as executed in each of these functions. An abstraction of these functions may be mapped to a generic model of software development, containing the overlapping phases depicted in figure 1 (top part), for each function (or process) of SPM. These phases will be supported by an agent framework. Figure 1 aims to explain the concept of this process of support. Figure 1 consists of the two basic concepts, namely the *phases of software development for each SPM key function* and the *software agent framework* that will address each key area of SPM individually.

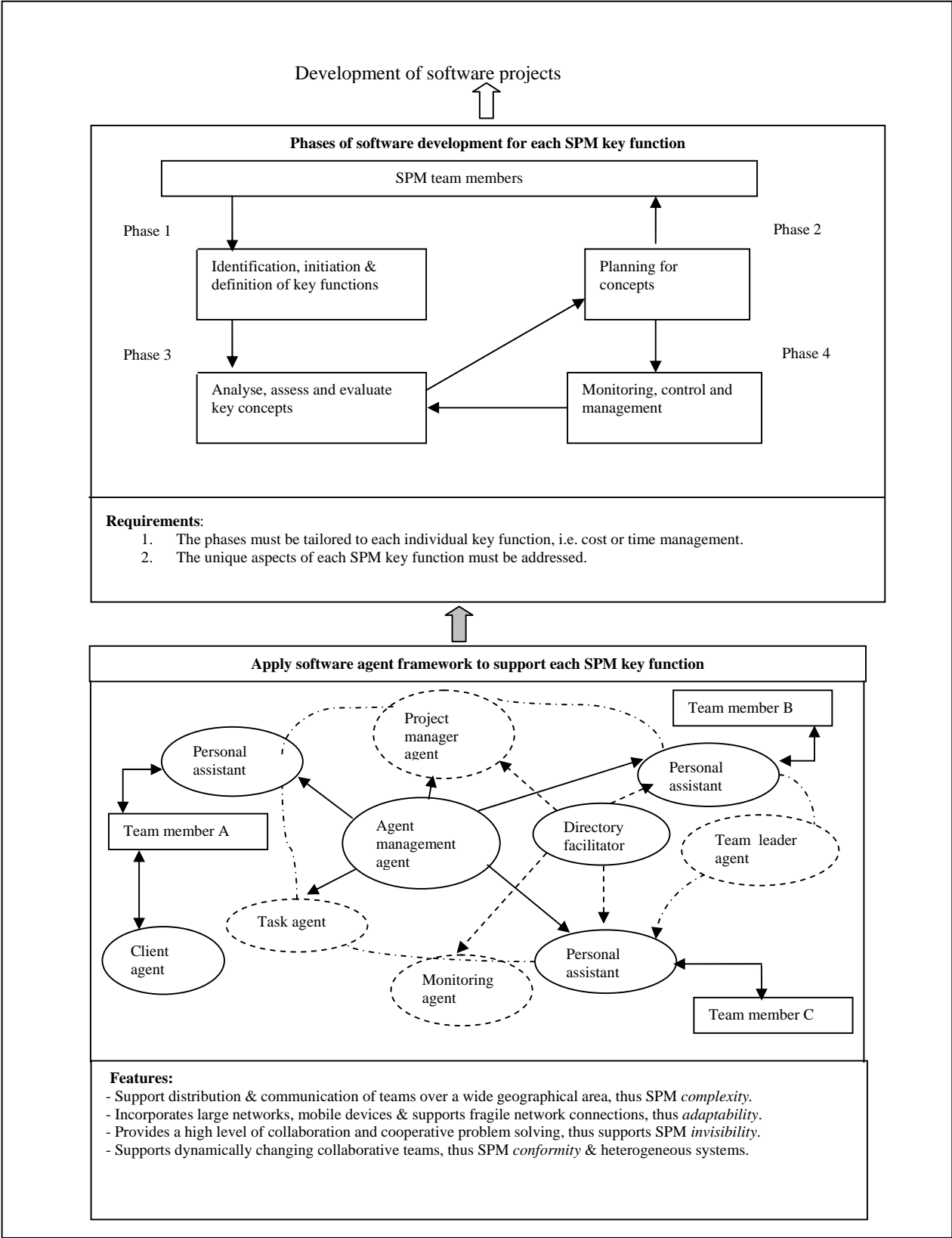


Figure 1 Conceptual view of the SPMSA model

The primary goal of the proposed SPMSA model is to support the teams and individual team members in the SPM environment while they are executing their tasks and, in so doing, to enhance the complete SPM environment. This support is enabled by an agent framework as depicted in figure 1 (bottom half). The team leader, teams and individual team members will be supported during each process of software project management, utilising an agent framework to simplify the environment, eliminate the complexities, enhance coordination and communication, implement dynamic changes in the system, support task scheduling and enhance all processes.

Table 3 provides a summary of the purpose of each agent (depicted in figure 3) forming part of the agent teams of the different SPM key functions of the SPMSA model.

Table 3 Agents and their tasks

Agents	Purpose	Mobile	Stationary
Agent management agent	<ul style="list-style-type: none"> • Manages the team of agents • Keeps track of the distribution location of all agents • Enables communication of agents • Enables mobility of agents • Tracks instantiation of tasks 		X
Client agent	<ul style="list-style-type: none"> • Executes a specialised task at a workstation • Interacts with the agent team • Receives input from task agent 		X
Directory facilitator	<ul style="list-style-type: none"> • Automated JADE facility • Provides Yellow Pages functionality to agents • Provides agents with information on services provided by other agents 		X
Personal assistant	<ul style="list-style-type: none"> • Allocated to a specific team member • Assists the team member • Interface between team member and other agents • Collaborative nature 		X
Messaging agent	<ul style="list-style-type: none"> • Traverses the network of agents • Carries messages to and from agents • Collaborates with agents 	X	
Monitoring agent	<ul style="list-style-type: none"> • Monitors agent movement • Monitors tasks and activities • Coordinates agents 	X	
Project manager agent	<ul style="list-style-type: none"> • Supports and directs the team of agents • Takes on project manager role • Helps create & initialise project • Specification of tasks • Allocation of tasks to members 	X	
Task agent	<ul style="list-style-type: none"> • Supports a specific task, i.e. information gathering, information distribution, information retrieval • Traverses the network of team members for input • Calculates various measures, such as probability of risk occurring, ROI, NPV • Gives feedback to personal assistant agents 	X	
Team leader agent	<ul style="list-style-type: none"> • Manages the team of agents 	X	

The phases of software development for each of the SPM key functions, as illustrated in table 1, were delineated and investigated in detail to compile the comprehensive SPMSA model. The SPMSA model comprises all processes of SPM as supported by an agent framework, which are illustrated in figures 2 and 3. The core functions and the facilitating functions are presented separately, owing to space limitations.

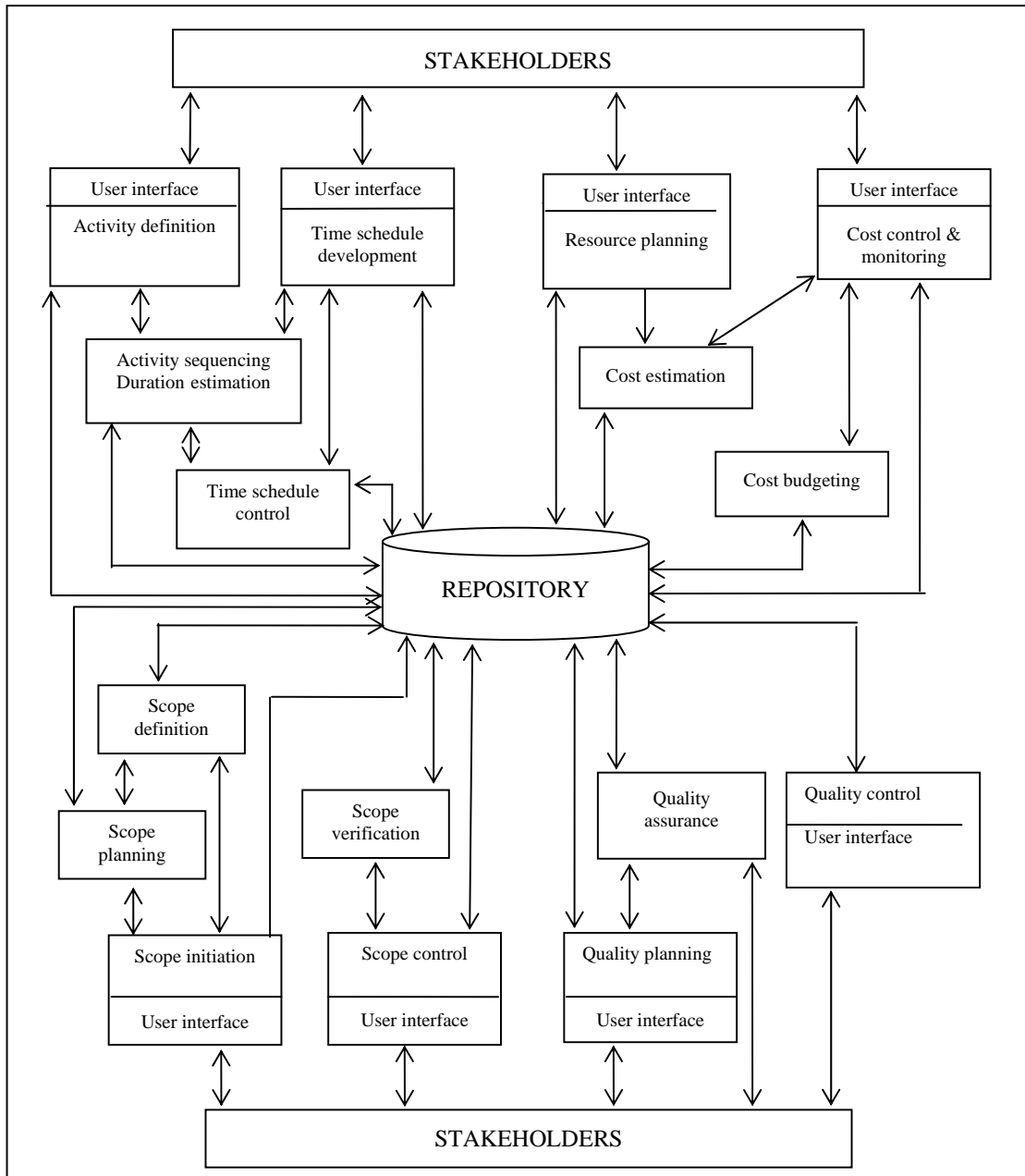


Figure 2 SPMSA model for core functions: scope, time, cost and quality management

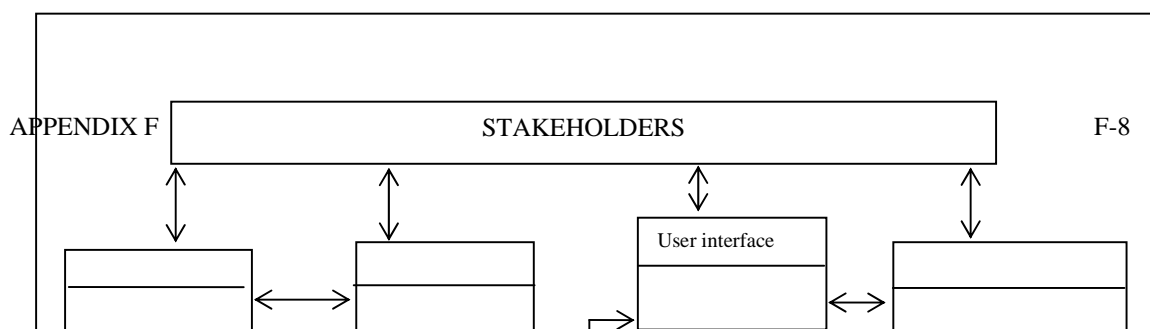


Figure 3 SPMSA model for facilitating functions: human resource, communication, risk and procurement management

The SPMSA model aims to enhance SPM processes by addressing the unique intrinsic aspects of SPM. The comprehensive framework of agents that forms part of the SPMSA model supports the entire SPM process, thereby aiming to eliminate failure and address shortcomings in this environment. This model will be unique in that it aims to support and enhance the *entire* environment of the SPM arena, and not only a section of it. Current software agent applications target only a section of this environment, for example planning or resource management. Unlike other programming paradigms, software agent technology not only provides support to the dynamically changing environment of SPM, but also support for the complex

heterogeneously distributed environment it encompasses. Furthermore, regular tasks may be automated and intelligence added to further support and enhance the workload of each team member.

4.2 Advantages of using agent technology for SPM

This study reveals that the limitations in the SPM environment can be supported by agent technology. A summary of the advantages of how the SPMSA model addresses the limitations of current SPM approaches through software agents is given in table 4.

Table 4 Limitations of SPM addressed by agent technology

Limitations of current SPM approaches	Agent enhancement
Environmental factors	
Stakeholders in virtual teams may have different goals and different backgrounds (Chen et al., 2003)	Virtual teams, supported by automated agent interaction, work toward a similar goal through coordination and collaboration of team members
Support communication in homogeneous environments and will need additional features or measures to connect heterogeneous elements (O'Connor & Jenkins, 1999)	Agents support heterogeneous environments, thus improving and enabling <ul style="list-style-type: none"> • communication • coordination
The systems are executed synchronously and must be connected to be executed (Maes, 1997)	Agent system executes asynchronously and autonomously, thus less network load & fewer communication overheads
Do not sufficiently support the knowledge representation of the SPM area (O'Connor & Gaffney, 1998)	Agent systems provide assistance in knowledge management, namely knowledge of plans and designs and mechanisms to reason on these elements
Human interaction/automated control	
Documents distributed by human action, thus allowing for human error, such as omission (Purvis, McCray & Roberts, 2000)	Automated workflow management to all relevant team members: <ul style="list-style-type: none"> • documents and information dispersed • documents and information retrieved from repository
Team member interaction dependent on human interaction, thus prone to errors (Petrie, Goldmann & Raquet, 2005)	Automates team member interaction: <ul style="list-style-type: none"> • regular prompting for input ensures that the data is current & tasks are not forgotten • improves productivity
All actions and coordination to be executed by team members without specific process coordination measures (Petrie et al., 2005)	Automates process coordination , which will improve programmer productivity as well as minimise errors
Tasks	
Complexity of tasks and environment is one of the reasons for failure (Benfield et al., 2006)	Complexity of tasks is minimised by automated support, such as automated calculations , thus reducing complexity of the solution and improving programmer productivity
Large systems are difficult to maintain consistently over set period of time. Current tools do not provide proper change notification (Petrie et al., 2005)	<ul style="list-style-type: none"> • Maintenance is automated and users are prompted for input on changes • Change control is automated and users are regularly prompted for input. Changes are incorporated dynamically
Current tools support reporting and calculation facilities, but not continuous progress management (Chandrashekar et al., 2002)	Management of progress status automated , e.g. risk monitoring and risk status checked on daily basis, enabling the project manager to identify problems early and take proactive measures (Roy, 2004)
Risks are commonly identified at the start of a project, but only 50% followed the risk through during development (Verner &	Agents monitoring risk will automate the continuous monitoring of risks , thus following all risks throughout the project

Limitations of current SPM approaches	Agent enhancement
Cerpa, 2005)	
Overemphasise passive corporative reporting aspects (Chen et al., 2003) of SPM by different team members	Continuous input of task status and sharing of information changes passive reporting to a system that supports dynamic reporting , improving coordination and cooperation between team members
Data, tasks and results will be sent over a network to execute at the user's workstation	Tasks embedded into agent behaviour, thus traversing the network agents, which lessens communication overheads and network load – tasks are executed at team member's site
Ineffective and inefficient communication, i.e. untimely information, failure to notify all team members (Chen et al., 2003)	Collaborative tool providing automated support on structures for efficient information sharing, set format for information storing and structures for communication will promote adequate and timely information sharing (Gawinecki, Kruszyk, Paprzycki & Ganzha, 2007)
Quality measures and standards selected by team. Human inaccuracy and omissions possible (O'Connor & Jenkins, 1999)	Continuous automated input of all team members regularly for quality control , as well as measures and directives conforming to standards
Current SPM tools provide no intelligent support on standards or best practices (O'Connor & Moynihan, 2000)	Agents with intelligence may encapsulate areas of experience, such as standards, and advise the project leader on best practices and standardisation, thus providing knowledge base support (Gawinecki et al., 2007)
Intelligent support	
Bidding and negotiation done by humans ((Badica, Popescu, Vukmirovic, Gawinecki, Kobzdej, Ganzha, & Paprzycki, 2007)	AI advantages of bidding and negotiating agents . Automation of these functions will mean less work and additional productivity for the developer /s
Current SPM tools that make projections concerning tasks and decisions are static and do not support dynamic simulation (Joslin & Poole, 2005)	Agent systems support dynamic simulation concerning planning of uncertainty, i.e. dynamic resource allocation. Simulations may help managers to anticipate critical conditions earlier and enable them to implement preventive measures; thus proactive SPM
All interaction through stakeholders but no support in decision making process of project manager (O'Connor & Gaffney, 1999; Purvis et al., 2003)	Personal assistant agent supports each individual team member and manages and analyses large amounts of project data (Gawinecki et al., 2007)

5. Verification of the SPMSA model

The SPM phases of the SPMSA model are compared with the processes in the ISO 10006:2003 standard (which targets projects specifically) to determine the relevance of the SPMSA model regarding software project management processes – see table 5. In table 5 the correlating phases are marked in the same colour, e.g. if the SPMSA model's phase correlates with clause 4: Quality management of ISO 10006:2003, both the SPMSA model's phase and the ISO clause are marked in red. Each similar set of processes is colour-coded (see colour code at the top of table 5).

The ISO 10006:2003 standard consists of a full and extensive list of clauses. Similar to ISO 9001:2000, the ISO 10006:2003 standard consists of 8 main clauses, 27 sub-clauses and 61 sub-sub clauses. Main clauses 1–3 concern only descriptive background, such as the scope of the document, normative references that state its correlation with ISO 9001:2000, as well as terms and definitions. These clauses are omitted from table 5 as they contain information and not processes to implement. In the standard, each first sub-clause, i.e. 1.1.1, 1.2.1 and 1.3.1, is a general clause, which is also omitted.

Table 5 SPMSA model vs ISO 10006:2003

Colour coding: all processes on: Quality management: red Scope management : turquoise Communication management: grey Review evaluations: green Time management: light blue Risk management: olive green Resource management: blue Cost management: orange Purchasing management: pink			
ISO 1006:2003 processes		SPMSA model	
Non-process clauses			
4 Quality management systems 4.1 Project characteristics 4.2 Quality management systems	4.1.2 Organisations 4.1.3 Processes in projects 4.1.4 Project management processes	Quality management	None
	4.2.1 Quality management principles 4.2.2 Project quality management 4.2.3 Quality plan for project	None	Quality assurance Quality control Quality planning
Process clauses			
5 Management responsibility 5.2 Strategic process 5.3 Reviews and progress evaluations			None Performance reporting (communication management)
6. Resource management	6.1.2 Resource planning 6.1.3 Resource control	Human resource management	Resource planning (cost management) Management: monitoring & control
6.2 Personnel-related processes	6.2.2 Establish project organisational structure 6.2.3 Allocation of personnel 6.2.4 Team development		Organisational planning Staff acquisition Team development
7 Product realisation 7.2 Interdependency-related processes	7.2.2 Project initiation & management plan development 7.2.3 Interaction management 7.2.4 Change management 7.2.5 Process and project closure		None None Change control (scope) Admin closure (communication)
7.3 Scope-related processes	7.3.2 Concept development 7.3.3 Scope development & control 7.3.4 Definition of activities None 7.3.5 Control of activities	Scope management	Initiation Planning Definition Verification Change control
7.4 Time-related processes	7.4.2 Planning of activity dependencies 7.4.3 Estimation of duration 7.4.4 Schedule development 7.4.5 Schedule control	Time management	Activity definition & sequencing Duration estimation Time schedule development

			Time schedule control None
7.5 Cost-related processes	7.5.2 Cost estimation 7.5.3 Budgeting 7.5.4 Cost control	Cost management	Cost estimation Cost budgeting Monitoring & control
7.6 Communication-related processes	7.6.2 Communication planning None 7.6.3 Information management 7.6.4 Communication control None None	Communication management	Identification & planning Team support Information distribution None Performance reporting (5) Admin closure (7.2.5)
7.7 Risk-related processes	7.7.2 Risk identification 7.7.3 Risk assessment 7.7.4 Risk treatment 7.7.5 Risk control None	Risk management	Risk identification Risk analysis & prioritisation Risk planning Monitoring Resolution
7.8 Purchasing-related processes	7.8.2 Purchasing planning & control 7.8.3 Documentation of purchasing requirements 7.8.4 Supplier evaluation 7.8.5 Contracting 7.8.6 Contract control	Procurement management	Procurement planning Solicitation planning Solicitation & source selection Contract administration Contract closure
8 Measurement analysis 8.1 Improvement processes	8.1 Improvement		None
8.2 Measurement & analysis	8.2 Measurement & analysis		None

All the key areas in the SPMSA model, namely scope management, time management, as well as cost, quality, human resource, communication, risk and procurement management, are reflected in the standard. ISO 10006:2003 also lists management responsibility (clause 5), product realisation (clause 7) and measurement analysis & improvement (clause 8) as separate clauses. The SPMSA model contains ten processes not reflected in the ISO model. The majority of processes in the SPMSA model correlate with processes in ISO 10006:2003. Furthermore, the eight core and facilitating functions of the SPMSA model are reflected in the ISO 10006:2003 standard.

The above comparisons clearly indicate that the SPMSA model conforms to the ISO 10006:2003 standard and, as such, can justifiably be applied to the software project management area. The SPMSA model addresses the shortcomings in current SPM applications, and the underlying technology, namely agent technology, will support the unique nature and changing environment of SPM.

6. Conclusion

It is clear from this research study that traditional methods and techniques of SPM do not meet the requirements posed by this dynamically changing and unique working platform. Software agent technology, although primarily applied to other fields, such as e-commerce, information retrieval and network management, is ideally suited to meeting the new challenges faced by the SPM characteristics, such as appropriate tools for effective sharing of information and knowledge among project contributors, as well as efficient distributed task scheduling, tracking and monitoring mechanisms. In this paper we propose an approach to using software agent technology to address these challenges. The SPMSA model was

compiled to enhance standard SPM practices and thus also to address challenges encountered due to the unique and changing environment of SPM. The SPMSA model is specifically tailored to address each of the unique features, namely *complexity*, *flexibility*, *conformity* and *invisibility*, through the agent framework. As is evident from this investigation, agent technology is extremely suitable for handling complex and dynamically *changing environments*. We believe that our solution is significant, based on our experience in other fields that advocate component-based development.

This research is aimed at software practitioners and software developers, but will also be beneficial to researchers working in the field of SPM. The development of software projects supporting crucial business activities may be utilised to attain a competitive advantage for that organisation.

7. References

- BADICA, C., POPESCU, E., FRACKOWIAK, G., GANZHA, M., PAPRZYCKI, M., SZYMCZAK, M. & PARK, M. 2008. On Human Resource Adaptability in an Agent-Based Virtual Organization. In: *Conference on Adaptive Networked Systems and Media, ANSYM 2008*. Wroclaw, Poland.
- BENFIELD, S., HENDRICKSON, J. & GALANTI, D. 2006. Making a strong business case for Multiagent Technology. In: *Proceedings of the AAMAS '06*. 1-59593-303 ed. Hokkaido, Japan, ACM Computing surveys. 8-12.
- BOEHM, B. W. 1991. Software Risk Management: Principles and Practices. *IEEE Software*, **8** (1), 32-41.
- BRAUN, P., EISMANN, J., ERFURTH, C. & ROSSAK, W. 2001. TRACY A Prototype of an Architected Middleware to support Mobile Agents. In: *Proceedings of the Eight Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems (ECBS'01)*. IEEE. **6**
- CHANDRASHEKAR, S., MAYFIELD, B. & SAMADZADEH, M. 1993. Towards automating software project management. *The International Journal of Project Management*. Elsevier Science Ltd. **11**(1), 29-39.
- CHEN, F., NUNAMAKER, J., F, ROMANO, N. C. J. & BRIGGS, R. O. 2003. A Collaborative Project Management Architecture. In: *Proceedings of the 36th Hawaii International Conference on System Sciences*. Big Island, Hawaii. IEEE.
- GAWINECKI, M., KRUSZYK, M., PAPRZYCKI, M. & GANZHA, M. 2007. Pitfalls of agent system development on the basis of a Travel Support System. In : W Abramowicz (ed), *Proceedings of the BIS 2007 Conference*, Springer, Berlin, LNCS 4439, 488-499.
- GELBARD, R., PLISKIN, N. & SPIEGLER, I. 2002. Integrating systems analysis and project management tools. *International Journal of Project Management*. Elsevier Science Ltd. **20** (6), 461-468.
- GREEN, S., HURST, L., NANGLE, B., CUNNINGHAM, P., SOMERS, F. & EVANS, R. 1997. *Software agents: a review*. URL: <http://www.cs.tcd.ie/Brenda.Nangle/iag.html> Accessed 20/05/2005.
- HUGHES, B. & COTTERELL, M. 2006. *Software Project Management*, Fourth Edition. London: McGraw-Hill. ISBN 10-0077-109899. 356.
- JENNINGS, N. R. 2001. An Agent-Based approach for building Complex Software Systems. *Communications of the ACM*, **44**, 35-39.
- JOSLIN, D. & POOLE, W. 2005. Agent-based simulations for software project planning. In: *Proceedings of the 2005 Winter Simulation Conference*. Orlando. IEEE Computer Society. 8.
- KOTZ, D. & GRAY, R. 1999. Mobile agents and the future of the Internet. *Operating systems review*, **33**, 7-13.
- LANGE, D. B. & OSHIMA, M. 1999. Seven good reasons for mobile agents. *Communications of the ACM*, **42**, 88-89.
- LETHBRIDGE, T. C. & LAGANIERE, R. (2001) *Object-oriented Software Engineering: Practical Software development using UML and Java.*, London: McGraw-Hill.

- MAES, P. Agents that reduce work and information overload. *Communications of the ACM*. **37**(7) 31-40.
- MARCHEWKA, J. T. 2003. *Information Technology Project Management*, Northern Illinois: Wiley and Sons. 319.
- O'CONNOR, R. & GAFFNEY, E. 1998. A Distributed Framework for implementing a Multi-agent Assistant System. Dublin, Ireland, Dublin City University. 1-12.
- O'CONNOR, R. & JENKINS, J. 1999. Using Agents for Distributed Software Project Management. *8th International Workshop on Enabling Technologies: Infrastructures for Collaborative Enterprises*. Stanford, USA. IEEE Computer Society Press. 54-60.
- O'CONNOR, R. & MOYNIHAN, T. 2000. An Agent Model of Decision Support for Software Project Management. *Advances in Decision Technology and Intelligent Information Systems*, **1**, 26-30.
- OGHMA: 2003. Types of Software Agents. *Oghma: Open Source*. URL: <http://www.oghma.org/>. Accessed 20/05/2003.
- OLSON, D. L. 2004. *Information Systems Project Management*, Nebraska, Second Edition. Boston:McGraw-Hill. 308.
- PETRIE, C., GOLDMAN, S. & RAQUET, A. 1999. Agent-based project managemnt. *Lecture Notes on Artificial Intelligence (LNAI)*, **1600**, 1-23.
- PURVIS, R. L., MCCRAY, G. E. & ROBERTS, T. L. 2003. The impact of Project Management Heuristics to IS Projects. In: *Proceedings of the 36th Hawaii International Conference on System Sciences*. Hawaii, IEEE. 1-7.
- ROY, G. G. 2004. A Risk Management Framework for Software Engineering Practice. In: *Proceedings of the Australian Software Engineering Conference (ASWEC'04)*. Australia, IEEE Computer Society.
- SCHWALBE, K. 2006. *Information Technology Project Management*, Canada: Thomson Course Technology.
- THE STANDISH GROUP INTERNATIONAL. 2005. Latest Standish Group Chaos Report. *Chaos Chronicles*. Massachusetts. URL: <http://www.standishgroup.com>. Accessed 04/03/05.
- VERNER, J. M. & CERPA, N. 2005. Australian Software Development: What Software Project Management Practices Lead to Success. In: *Proceedings of the 2005 Australian Software Engineering Conference (ASWEG)*. IEEE Computer Society Press.
- WOOLDRIDGE, M. 2002. *MultiAgent Systems*, West Sussex, England: John Wiley & sons, Ltd.ZANONI, R. & AUDY, J. L. N. 2003. Project Management Model for Physically Distributed Software Development Environment. In: *Proceedings of the 36th Hawaii International Conference on System Sciences (HICSS'03)*. Big Island. Hawaii. 249 (1-7).