



University of South Australia

A Framework for Steering Software-Intensive Acquisitions to Success

RICARDO P. PECULIS

BE (EESC/USP, Brazil) 1978

Computer Systems Engineering Centre
School of Electrical and Information Engineering
University of South Australia

A thesis submitted in fulfilment
of the requirements for the degree of
Doctor of Philosophy in Computer Systems Engineering

December 2011

Table of Contents

Table of Contents	i
List of Figures.....	v
List of Tables	vi
Glossary	vii
Acronyms and Abbreviations.....	xi
Summary.....	xiii
Declaration.....	xv
Acknowledgements.....	xvii
Addendum.....	xix
Chapter 1: Introduction	1
1.1. Background and Motivation	1
1.2. The Challenge of Software Development.....	1
1.3. Engineering Software-intensive Systems	4
1.4. Significance of the Problem.....	4
1.4.1. Failure of Software Projects.....	6
1.4.2. Failure of Software Projects in Australia	7
1.5. Objectives	8
1.6. Research Problem	8
1.6.1. Sub-Problem 1.....	9
1.6.2. Sub-Problem 2.....	10
1.6.3. Sub-Problem 3.....	10
1.7. Research Approach	10
1.8. Overview of this Thesis	11
Chapter 2: Literature Review.....	13
2.1. Success and Failure of Software-Intensive Projects	13
2.1.1. Success of Software-Intensive Acquisitions	14
2.1.2. Cobb's Paradox	15
2.1.3. The First Clue.....	16
2.1.4. The Importance of Knowledge.....	17
2.1.5. The Second Clue	18
2.2. The System of Profound Knowledge	19
2.2.1. Appreciate the System.....	20
2.2.2. Knowledge about Variation	21
2.2.3. The Third Clue	24
2.2.4. Knowledge of Psychology	24
2.2.5. Theory of Knowledge	25
2.3. Systems, System Thinking and Complex Adaptive Systems	26
2.3.1. Systems Theory and Cybernetics	27
2.3.2. Systems Thinking.....	32

2.3.3.	Complex Systems	35
2.3.4.	Structure Influences Behaviour	39
2.3.5.	Complex Adaptive Systems	40
2.3.6.	Framework for Harnessing Complexity	45
2.4.	Psychology and Social Behaviour.....	47
2.4.1.	Cognition, Emotion Perception and Motivation.....	49
2.4.2.	Theory of Motivation	49
2.4.3.	Theory of Behaviour.....	51
2.4.4.	Personality Types	53
2.4.5.	Groups, Teams and Organisations.....	55
2.4.6.	Social Behaviour in Groups, Teams and Organisations	58
2.4.7.	Knowledge and Learning	62
2.5.	Social Systems of Engineering Projects.....	66
2.6.	Summary	68

Chapter 3: Appreciate the Software-Intensive Acquisition System71

3.1.	System in Context	71
3.2.	Capability Development and Acquisition Processes.....	72
3.2.1.	Life Cycle Models	73
3.2.2.	The Role of Systems Engineering	75
3.3.	The Conceptual Acquisition Process	77
3.3.1.	Transformations through Domain Spaces	78
3.3.2.	Products and Tasks in the Acquisition Space	81
3.3.3.	Effectiveness, Efficiency and Performance.....	83
3.4.	Knowledge and Learning in Software-Intensive Acquisitions.....	84
3.4.1.	Management Knowledge.....	84
3.4.2.	Technical Knowledge	86
3.4.3.	Learning Processes and Behaviours	87
3.5.	The Software-Intensive Acquisition as CAS	89
3.5.1.	Adaptive Agents	91
3.6.	Emergent Properties	95
3.6.1.	Quality as an Emergent Property.....	96
3.6.2.	Knowledge as an Emergent Property	104
3.6.3.	Competency as an Emergent Property.....	108
3.6.4.	The Ultimate Emergent Property.....	109
3.6.5.	Undesirable Emergent Properties	109
3.7.	Systemic Structure	109
3.7.1.	Technical System	110
3.7.2.	Social System	111
3.7.3.	Underlying Processes	112
3.7.4.	Good and Bad Structures.....	114
3.8.	Hypotheses about the System.....	115
3.8.1.	First Hypothesis.....	115
3.8.2.	Second Hypothesis	116
3.8.3.	Testing the Hypotheses.....	117
3.9.	Summary	117

Chapter 4: Understanding Variation in the Software-Intensive Acquisition System....119

4.1.	Observed Variation in Software-Intensive Projects	119
4.1.1.	Observed Variation in Quality.....	121
4.1.2.	Observed Variation in Productivity	122
4.1.3.	Observed Variation in Schedule	123
4.2.	Causes of Variation in Software Development	124
4.3.	A Case of Quasi-Perfect Software	126
4.4.	Variation in Software-Intensive Acquisitions	129

4.4.1. Common and Special Causes of Variations	130
4.5. Variability, Uncertainty and Risk	137
4.6. Hypotheses about Variation.....	139
4.6.1. Third Hypothesis.....	139
4.6.2. Fourth Hypothesis	140
4.6.3. Testing the Hypotheses	141
4.7. Summary	141
Chapter 5: The Software-Intensive Acquisition Model.....	143
5.1. A Case for Modelling and Simulation	143
5.2. Modelling and Simulation applied to Social Systems	145
5.2.1. System Dynamics Models.....	146
5.2.2. Agent-Based Models	146
5.2.3. Verification and Validation of Social Computational Models	148
5.3. Objectives	149
5.4. Theory and Hypotheses	150
5.4.1. Complex Adaptive System Hypothesis.....	150
5.4.2. Problem-Solving Group Hypothesis	151
5.4.3. Non-Determinism Hypothesis.....	151
5.4.4. Ideal System Hypothesis	151
5.5. Software-Intensive Acquisition Agent-Based Model	152
5.5.1. ACTS Theory	154
5.6. Task Model	157
5.6.1. Virtual Products and Tasks in the Acquisition Space	158
5.6.2. Chinese Whispers Effect	162
5.7. Cognitive Agent Model	162
5.7.1. Perception and Expression Models	164
5.7.2. Social Model	164
5.7.3. Task Assignment.....	166
5.7.4. Emotion Model.....	167
5.7.5. Cognition Model	169
5.7.6. Physics Model	171
5.7.7. Behavioural Model.....	171
5.8. Social Environment Model	173
5.8.1. Relationship Model	175
5.9. Modelling Variation.....	178
5.10. Summary	179
Chapter 6: Exploration in a Simulated Environment	181
6.1. Overview.....	181
6.1.1. A Brief Description of the Simulation Process	181
6.2. Task Configuration	186
6.3. Promising Results	192
6.3.1. Influence of Knowledge and Interaction Styles	192
6.3.2. Influence of Learning through Collaboration.....	195
6.4. Testing Hypotheses	196
6.4.1. Ideal Scenario.....	196
6.4.2. First Hypothesis	201
6.4.3. Second Hypothesis	207
6.4.4. Third Hypothesis	212
6.4.5. Fourth Hypothesis	212
6.5. Summary	223
Chapter 7: Conclusions	225
7.1. Introduction.....	225

7.2. The Research.....	226
7.3. Answer to the Research Sub-Problems	228
7.3.1. Answer to Sub-Problem 1	228
7.3.2. Answer to Sub-Problem 2	229
7.4. Answer to the Research Problem	230
7.5. The Need for a Framework	231
7.6. A Framework for Steering Software-Intensive Acquisitions to Success	232
7.6.1. The System and its Aims	234
7.6.2. The Process of Knowledge Management.....	235
7.6.3. The Process of Behaviour Management.....	236
7.6.4. The Process of Motivation Management.....	238
7.6.5. Emergent Results.....	239
7.7. Limitations of this Research.....	241
7.8. Future Work	241
7.8.1. Validate SIAABM and SIAABMSim	242
7.8.2. Explore, Refine and Expand SIAABM and SIAABMSim.....	242
7.8.3. Validate the Effectiveness of the Proposed Framework.....	243
7.9. Final Remarks	244
Bibliography	245
Appendix A: Simulation Environment	261
Appendix B: Refereed Paper 1	283
Appendix C: Refereed Paper 2	299
Appendix D: Refereed Paper 3	307
Appendix E: Refereed Paper 4	315
Appendix F: Refereed Paper 5.....	325
Appendix G: DVD Contents.....	333

List of Figures

Figure 1 – Transformations between Domain Spaces	79
Figure 2 – Actual transformations between Domain Spaces.....	80
Figure 3 – Useful Knowledge.....	161
Figure 4 – Cognitive Agent Architecture	164
Figure 5 – Actor Behavioural Styles.....	168
Figure 6 – Actor’s Knowledge Profile.....	170
Figure 7 – Actor’s Behavioural Model	172
Figure 8 – Social Environment model	174
Figure 9 – Relationship Model	175
Figure 10 – Feedback Loops.....	178
Figure 11 – Duration, Experience and Interaction Styles.....	194
Figure 12 – Schedule Performance	198
Figure 13 – Schedule and Cost Performance	199
Figure 14 – Effectiveness 3 Organisations Waterfall Development.....	200
Figure 15 – Effectiveness 3 Organisations Incremental Development.....	200
Figure 16 – Organisations File - 3Orgs-B017-K03-E03-M10-SD005-750.....	201
Figure 17 – Probabilities File.....	202
Figure 18 – Effectiveness x Knowledge Gap and Behavioural Style	203
Figure 19 – Schedule Performance (Ps) x Knowledge Gap and Behavioural Style....	204
Figure 20 – Cost Performance (Pc) x Knowledge Gap and Behavioural Style.....	205
Figure 21 – Efficiency x Knowledge Gap and Behavioural Style.....	206
Figure 22 – Effectiveness x Knowledge Gap and Behavioural Style.....	208
Figure 23 – Schedule Performance (Ps) x Knowledge Gap and Behavioural Style....	209
Figure 24 – Cost Performance (Pc) x Knowledge Gap and Behavioural Style.....	210
Figure 25 – Efficiency x Knowledge Gap and Behavioural Style.....	211
Figure 26 – Organisations’ File for testing the Fourth Hypothesis	213
Figure 27 – Fourth Hypothesis: Non-Collaborative Performance and Behaviour	218
Figure 28 – Fourth Hypothesis: Collaborative Performance and Behaviour.....	220
Figure 29 – Motivation, Behaviour and Action	240
Figure A-1 – SIAABMSim SDE	261
Figure A-2 – SIAABMSim Main Screen	262
Figure A-3 – Environment Window	263
Figure A-4 – Artefact Effectiveness Colour Code.....	264
Figure A-5 – Environment Window – Artefacts View.....	265
Figure A-6 – Environment Window – Artefacts and Dependencies Views	266
Figure A-7 – Actor Role Symbols	267
Figure A-8 – Environment Window – Actors and Interactions Views.....	268
Figure A-9 – Behaviour Window	269
Figure A-10 – Performance Window.....	269
Figure A-11 – Simulation Parameters	271
Figure A-12 – Organisations File	272

Figure A-13 – Organisations File Format	273
Figure A-14 – Probabilities File.....	274
Figure A-15 – Transformations File.....	277
Figure A-16 – Transformation File	277
Figure A-17 – Changing Attributes of an Actor during the Simulation.....	278
Figure A-18 – SimSettings-Ideal and SimSettings-Real.....	279
Figure A-19 – Probabilities-Ideal and Probabilities-Real	280
Figure A-20 – Organisations-Ideal.....	281

List of Tables

Table 1 – Senge’s Five Disciplines	34
Table 2 – Five Disciplines mapped to Deming’s System of Profound Knowledge.....	34
Table 3 – Seven Basics Characteristics of CAS	40
Table 4 – LSI Attributes.....	54
Table 5 – Five Orders of Ignorance	63
Table 6 – Software-Intensive Acquisitions Seven Basics Characteristics of CAS	90
Table 7 – Sub-Set of Rules in an Engineer Agent Performance System	92
Table 8 – Another Sub-Set of Rules in an Engineer Agent Performance System	93
Table 9 – Software Development Activities for Large Military Projects	120
Table 10 – Average Number of Delivered Defects per Function Point.....	121
Table 11 – Productivity Probability Ranges	123
Table 12 – Software Schedule Ranges in accordance with Project Size	124
Table 13 – Cost Comparison of the Space Shuttle Onboard Software	127
Table 14 – SIAABM Conformance to the Important Characteristics of ABM	153
Table 15 – SIAABM Conformance with the ACTS Theory Agent Axioms	155
Table 16 – SIAABM Conformance with the ACTS Theory Interlink Axioms	156
Table 17 – The Fundamental Scale of AHP.....	159
Table 18 – Actor Role Profile	165
Table 19 – Probability of Action in accordance with the Actor’s Behavioural Style...169	169
Table 20 – Actor Specialty Profile.....	170
Table 21 – Aircraft Roles	187
Table 22 – The Need Transformation Matrix (T1)	187
Table 23 – Observed Variations of Effectiveness, Cost and Schedule	215
Table 24 – Fourth Hypothesis: Number of Constructive and Knowledgeable Actors..217	217
Table 25 – Fourth Hypothesis: Variation in Actor’s Attributes.....	221
Table A-1 – Events in the Probabilities File	275
Table A-2 – SimSettings File Description	276

Glossary

Term	Definition
Acquisition	Acquisition is the process of acquiring products by specification, contract and eventual development.
Actor	Actor is a cognitive agent that represents people in an agent-based model and simulation.
Artefact	Artefact is a non-cognitive agent that represents objects in an agent-based model and simulation.
Behaviour	Behaviour, in the context of this research, is any noticeable change, response or action of a person, a virtual agent or system. Behaviour is driven by motivation and results into action.
Capability	Capability is the ability to achieve a particular operational effect.
Causal Analysis	The analysis of defects to determine their cause (SEI 2002, Appendix A, Glossary of Terms, p. A-3)
Chinese Whispers	'Chinese Whispers' is used as a metaphor for cumulative error when information is transmitted from one recipient to another. The metaphor is based on a game Chinese Whispers, also known as Telephone, Grapevine, Broken Telephone and Whisper Down the Lane, where the first player whispers a phrase or sentence to the next player. Each player successively whispers what that player believes he or she heard to the next. The last player announces the statement to the entire group. Errors typically accumulate in the retellings, so the statement announced by the last player differs significantly, and often amusingly, from the one uttered by the first. Source Wikipedia, http://en.wikipedia.org/wiki/Chinese_whispers .
Cognition	Cognition refers to activities as thinking and reasoning and is associated with any class of mental 'behaviours' leading to the mental picture of ideas, pieces of knowledge, complex concepts and problem solving. Adapted from the Dictionary of Psychology (Reber & Reber 2001).
Complex Adaptive System	Complex Adaptive System (CAS) is an aggregate of independent agents (entities) with the capacity to modify their rules of behaviour in response to stimuli created by interacting with other agents and with the environment. The change of behaviour, named 'adaptation', produces collective unpredictable behaviour as emergent properties of the system (extracted from Holland 1995).
Culture	Culture, in the context of this research, is the system of information that codes the manner in which the people in an organised group, society or nation interact with their social and physical environment (from Reber & Reber 2001).
Data	Data, in the context of this research, is the body of evidence or facts gathered in an experiment or study (from Reber & Reber 2001).
Effectiveness	Effectiveness indicates how well the system is prepared to achieve its objective.
Efficiency	Efficiency indicates how much effort or energy could be wasted for a system to achieve its objective at its nominal effectiveness.
Epistemology	Epistemology is a theory concerning the means by which we may have, and express, knowledge of the world (Checkland1993 p. 314).
Emergence	Emergence is a characteristic of systems that exhibit new properties and

Term	Definition
	behaviour as a result of the interaction of its components. Emergent properties do not exist in any of the system's component in isolation or before the system comes into existence.
Emotion	Emotion is an umbrella term for the state created by feelings such as love, hate, fear, terror, good and bad feelings, etc. Emotional state is not the result of rational thinking. Feelings, as one of the dimensions of emotion, can lead to hypothesis developing beliefs that are not supported by real evidence. Adapted from the Dictionary of Psychology (Reber & Reber 2001).
Experience	Experience, in the context of this research, is the knowledge gained by a person from participating in an event (Reber & Reber 2001). Experience is gained when a person puts in practice the knowledge he or she already has. Experience allows a person to develop mental models of events not yet experienced. Specific experience is experience of a specific field.
Expertise	Expertise, in the context of this research, is knowledge of a particular field possessed by a person.
Extrinsic Motivation	Extrinsic motivation is a construct that pertains whenever an activity is done in order to attain some separate outcome (Ryan & Deci 2000 p. 60).
Framework	Framework is a structure of principles, assumptions, concepts, values, and practices that constitutes a way of viewing reality.
Holism	Holism when applied to systems is a paradigm concerned with wholes and their properties (Checkland 1993 p. 13). The holistic paradigm holds that investigation and analysis of a system cannot be reduced to the analyses of its components without considering how they are connected and interrelated.
Information	Information, in the context of this research, is the term used to quantify an array of items in terms of the number of choices one has in dealing with them (from Reber & Reber 2001). Information is data placed into a context.
Intrinsic Motivation	Intrinsic motivation is defined as the doing of an activity for its inherent satisfactions rather than for some separable outcome (Ryan & Deci 2000 p.56).
Knowledge	Knowledge, in the context of this research, is the body of information possessed by a person or, by extension, by a group of persons (adapted from Reber & Reber 2001). Knowledge is information placed into a context. Specific knowledge is knowledge of a specific field. Explicit knowledge is the knowledge that can be documented and stored in some form of media. Implicit knowledge is knowledge that is not documented in any form of media.
Learning	Learning, in the context of this research, is the process of acquiring knowledge through individual effort, formal training, collaboration and experience.
Model	Model is a representation used to communicate and facilitate understanding of complex entities or systems. Models can be used to experiment with how entities and systems react under specific conditions and assist in predicting the behaviour of the real system. Computational models are constructed as computer programs.
Motivation	Motivation, in the context of this research, is the intervening process or internal state of a person or a virtual agent that drives their behaviour or impels them into action. (Adapted from Reber & Reber 2001).
Organisation	A system in which people are the most numerous part and which is so organised as to create products and/or services of value to others (Rechtin 2000).
Perception	Perception, in the context of this research, comprises those processes that

Term	Definition
	give state and unity to sensory input; it is the person's interpretation of a situation, bringing awareness of what the person believes is the truth of something. (Adapted from Reber & Reber 2001).
Psychology	Psychology, in the context of this research is what scientists and philosophers of various persuasions have created to try to fulfil the need to understand the minds and behaviours of human beings as individuals. (Adapted from Reber & Reber 2001).
Reductionism	Reductionism is an approach to scientific investigation and analysis of complex problems (and systems) by breaking them into their more treatable components. Reductionism can be seen as the opposite of holism.
Simulation	Simulation is a replication of a condition applied to an artefact, entity or system using a model or artificial environment. It is the process to configure or stimulate models to experiment with how they react under specific conditions to assist in predicting the behaviour of the real system. Computer simulation uses computers to simulate computational models.
Skill	Skill is the ability to successfully apply knowledge to execute a task and achieve an intended goal. (Adapted from Reber & Reber 2001).
Software Community	All people and organisations involved in the research, acquisition, specification and development of software.
Software-intensive Capability	Software-intensive capability is a software-intensive system used to achieve a particular operational effect.
Software-intensive System	Software-intensive systems are technical systems with a strong dependency on specific and innovative software to execute their functions. Without software a software-intensive system becomes inoperative.
Software-intensive Project	Software-intensive projects develop software-intensive systems through the application of engineering disciplines.
Software-intensive Acquisition	Software-intensive acquisition is the process of acquiring (see Acquisition) software-intensive products intended to operate as part of software-intensive systems.
System	System is a network of interdependent components that work together to try to accomplish the aim of the system (Deming 2000 p. 50)
Variance	Variance is the distance between the mean of a set of data to any point in the data.
Variation	Variation is the difference between what is expected and what is observed. Numerically, variation is the amount of the difference between an expected (or planned) value and the actual value.
Variety	The total number of possible states of a system, or of an element of a system (Beer 1972).

Acronyms and Abbreviations

Acronym	Definition
ACM	Association for Computing Machinery
ACTS	Agent, Cognition, Task and Social environment
ASDEFCON	Australian Standard Defence Contracting
BCAG	Boeing Commercial Airplane Group
BMP	Behaviour Management Plan
CAS	Complex Adaptive System
CCPM	Critical Chain Project Management
CMM	Capability Maturity Model
CMS	Configuration Management System
COTS	Commercial-off-the-shelf
CSC	CSC is a global business solutions and technology services provider. CSC, formerly CSA (Computer Sciences of Australia), was formed in 1970 as a joint venture of the AMP Society and CSC of the USA, becoming totally owned by AMP from 1988 until 1994, whereat it was wholly acquired by CSC becoming CSC Australia. CSC Australia was involved in several Australian Defence projects including the Collins Class Submarine, the Combat System Simulator and Trainer for the ANZAC class frigates and the Super Seasprite Helicopter.
DAF	Department of Air Force (USA)
DCDM	Defence Capability Development Manual
DMO	Defence Materiel Organisation (Australia)
DOD	Department of Defence
DPPM	Defence Procurement Policy Manual
EESC	Escola de Engenharia de São Carlos (Portuguese for School of Engineering of Sao Carlos)
FLOPS	FLoating point OPerations per Second
FP	Function Point
FPGA	Field Programmable Gate Array
FSF	Free Software Foundation
GSI	Group Styles Inventory
GAO	(US) Government Accountability Office
GNU	General Public License
HCI	Human-Computer Interface
IT	Information Technology
ICD	Interface Control Document
KMP	Knowledge Management Plan
KTS	Keirsey Temperament Scale
LSI	Life Style Inventory
MBTI	Meyer-Briggs Type Indicator

Acronym	Definition
NATO	North Atlantic Treaty Organisation
OCI	Organisational Culture Inventory
PMI	Project Management Institute
R&D	Research and Development
RAAF	Royal Australian Air Force
ROI	Return on Investment
RVG	Radar Video Graphics
SCD	Specification Control Document
SEI	Software Engineering Institute
SIAABM	Software-Intensive Acquisition Agent-Based Model
SIAABMSim	Software-Intensive Acquisition Agent-Based Model Simulation
SLOC	Source Lines of Code
SPC	Statistical Process Control
OS	Operating System
OSI	Open Source Initiative
PMBOK	Project Management Body of Knowledge
SR&O	System Requirements and Objectives
ULS	Ultra-Large-Scale
USP	Universidade de São Paulo (Portuguese for University of Sao Paulo)

Summary

Abstract: Achieving the established parameters of cost, schedule and quality in software-intensive acquisitions is a challenge for engineering and management. Many approaches have been attempted without producing consistency of results and failure is still the norm. This thesis researches root causes of success and failure of software-intensive acquisitions by examining the acquisition as a system, variation in the system, and people's attributes and social behaviours, individually and in groups that influence the acquisition as a whole. A complex agent-based model focused on the engineering process is applied to test various hypotheses about the system and its variations. The results from the research provide a framework for steering, rather than attempting to control, software-intensive acquisitions to success.

Aims: The objective of this research is to develop a better understanding of what drives the success or failure of software-intensive acquisitions and find ways to assist managers and decision makers make better decisions when faced with the challenges of the complex environment of software-intensive acquisition.

Method: Deming's System of Profound Knowledge guides this research in appreciating software-intensive acquisitions as a socio-technical system; understanding the variations present in the system; and understanding people and how they behave in a social system. To test the hypotheses about the system and its variations a complex agent-based model, (a model of the models), has been developed that groups agents into various organisational hierarchies working in an over arching engineering life cycle. The model also takes account of different individual and team behaviours, skills, learning abilities, forgetfulness, and formal and informal relationships between actors and task assignments. The use of the simulation tool that implements the model helps formulate a framework for steering software-intensive acquisitions towards intended goals. Various intuitive or commonly believed results are demonstrated using as supporting information, public information on a major software-intensive acquisition Defence project.

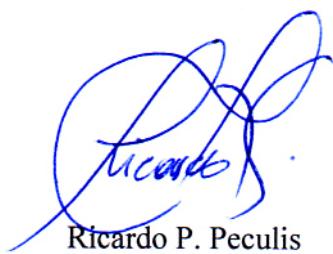
Results: The research confirms, for example, that:

- 1) Distortions caused by lack of knowledge and experience propagate through intermediate products into the final product – creating a *solution* that will not satisfy a user's *need*;
- 2) Individual variation brought about by different behaviours and attitudes is the fundamental cause of variations to cost, schedule and quality;
- 3) Failure occurs when the intrinsic aim of the system is not aligned with the intended aim for the system; and
- 4) Learning behaviours and constructive environments help to align the two aims and steer the acquisition towards its intended goals.

Conclusions: The model developed in this thesis allows an exploration of real-world factors with software-intensive acquisitions and provides new understanding about how to create the necessary conditions that will contribute to project success and why this is so. The original contribution to knowledge of the thesis is in: (i) the complexity of the agent-based model in bringing together a number of models from different fields relevant to the problem space; (ii) confirming widely held beliefs about why major projects succeed or fail, helping to validate the research; and (iii) supporting a change in focus from controlling complex software-intensive acquisitions to guiding them – planning for the journey rather than a detailed planning of the journey.

Declaration

I declare that this thesis presents work carried out by myself and does not incorporate without acknowledgement any material previously submitted for a degree or diploma in any university; and to the best of my knowledge it does not contain any materials previously published or written by another person except where due reference is made in the text.



Ricardo P. Peculis



Acknowledgements

I am pleased to acknowledge the support I received from CSC Australia during this research and for the opportunity I had to experience the true nature of software-intensive projects. My thanks go to my peers and friends Neil Finlayson, Peter Morgan, Robert Phillips, John Bocking and Narayan Sundararajan, for the many insightful discussions; and to George Bourne for being bold and allowing me to apply concepts derived from this work in his projects.

I am grateful to my supervisors Dr Derek Rogers and Professor Peter Campbell for their assistance and encouragement. Their reviews and feedback were deeply appreciated.

My acknowledgements would not be complete without including my family: my parents José and Zenaide, for their support and without whom I would not be who I am; my lovely wife and best friend Loana, for always being with me all the way; my son Rubens, for the many reviews of my drafts and who so patiently listened to my passionate brainstorm sessions; my daughter Luíza and my son Caio, for their encouragement. To them I offer my eternal gratitude.

Finally, I have to acknowledge the contribution of so many managers, engineers and software developers I had the privilege to work with throughout my career and from whom I learned so much. Their dedication, enthusiasm and occasional frustration taught me that our individual best is not always enough and motivated me to go searching for better answers.

Addendum

One of the thesis examiners raised the following questions and observations:

- a. *What specific insights were developed as result of developing and using the SIAABMSim simulation and model?*
- b. *Why were these insights not obvious without the model – are any of the results counter-intuitive?*
- c. *Was it obvious to begin with by way that the model was structured that these hypotheses would be confirmed even without the need to actually run de model?*
- d. *The model by Mr. Peculis represents one of the many possibilities for a quantitative model that fits the qualitative theories ..., i.e., which of many possible quantitative models truly or best represent the system being modelled in terms of its predictive capabilities?*
- e. *If the model necessarily confirms the kind of behaviours observed in real software-intensive acquisitions, then this part of the work should be regarded as calibration of the framework rather than a validation of either model or the theories that comprise it.*

In response to the examiner's questions and observations the author offers the following response:

The author acknowledges that the model proposed by this thesis is one of many and confirms the observation in (d). The model, as stated in section 5.3, is not intended to be a predictive tool, but one that helps to develop a better understanding of some aspects of complex software-intensive acquisitions. Therefore, finding the best predictive model, if possible at all, is beyond the scope of this research and should be an area of future work.

In response to (c), the research searched reputable theories that would support the author's observations resulting from his professional experience. As a consequence the model was structured based on the author's experience and applying theories that would

support it.

Regarding the insights developed by the research, questioned in (a) and (b), the author highlights what was learned about the impact of individual attributes of cognition and behaviour and social structures that create the conditions the cause one individual to influence another, and how these influence the software-intensive acquisition as a whole. Whether these insights would be obvious without the model is a question for the reader. What is obvious for some may not be so for others. The evidence of failure of software-intensive acquisitions supports the case that the cognitive and social aspects highlighted by this research are not always taken into consideration and it is an indication that the insights obtained from the model are not obvious to many managers and decision-makers.

Complementing the response to (a), the author recalls the final remarks of this thesis stating that the process of constructing a model and simulation of complex system is a rich experience that brings a wealth of insights, knowledge and consequent understanding.

Finally, answering the observation in (e), the author acknowledges that the simulation results from the application of the proposed model are in agreement with the author's experience and confirm the kind of behaviours observed in real software-intensive acquisitions. The calibration of the framework and specific facsimile models for real projects should be regarded as future work, as proposed by the author in section 7.8.

Chapter 1: Introduction

1.1. Background and Motivation

The need for software continues to grow and electronic gadgets, telecommunications, business and defence systems increasingly cannot operate without software or in the presence of software failure. The pressure to build more complex systems in short time spans and at competitive cost, make software development one of the major challenges in the success of acquisition of new software-intensive systems. Software-intensive systems are technical systems with a strong dependency on specific and innovative software to execute its function. Acquisition is the process of acquiring products by specification, contract and eventual development. Therefore, software-intensive acquisition is the process of acquiring software-intensive products intended to operate as part of software-intensive systems.

The author's personal motivation to develop this research comes from working on engineering projects for many years. The author's experience shows that success in projects with large software components is the result of many factors, some hidden or completely unknown, and beyond the capacity and effort of individuals in isolation. The hard lesson learned from experience is that in these kinds of projects the best intentions of individuals may not be enough. This research is the author's personal endeavour to explain the perceptions he developed through his professional experience in searching for answers to the question: *Why is it so difficult to succeed in large software-intensive projects?*

1.2. The Challenge of Software Development

Software is the result of a creative process, where collaboration and cooperation between customer and developer are essential. Software does not exist by itself and is a replication of the solution given to a problem, which is not always defined and resolved before the software development starts. Humphrey uses the definition of 'knowledge work' (Drucker in Humphrey 2010a) to describe software as a 'different kind of work', being 'work with the mind rather than with the hands', where 'the products, instead of being things you can touch and feel, are ideas' (Humphrey 2010a p. 4).

The US Department of Air Force in their ‘Guidelines for Successful Acquisition and Management’ (DAF 2000) and the US General Accounting Office (GAO) in their 2004 report on Software Intensive Weapon Acquisitions (GAO 2004) highlight that schedule delays, cost overruns and systems with reduced performance are notorious in software-intensive acquisitions. The Standish Group reported in its first CHAOS Report (Standish Group 1994) that the main reasons that cause failure and ultimately the cancellation of software projects are lack of user involvement and incomplete requirements and specifications. In the same report the Standish Group also reported that user involvement, executive management support and a clear statement of requirements, top the list of factors that contribute to project success. In the subsequent reports (Standish Group 1995, 1999, 2001, 2003, 2005, 2009) stopped reporting the causes of failure of software projects but continued consistently reporting that the two main factors that contribute to success are user involvement and executive management support. A clear statement of requirements or clear business objectives featured as the third important factor for achieving success, except in the 2001 report (Standish Group 2001), where clear business objectives was in fourth place and firm basic requirements was in seventh place. Although the validity of the reports published by the Standish Group has been challenged by Glass, Eveleens and Verhoef (Glass 2005, 2007; Eveleens & Verhoef 2010), their qualitative findings confirm what has been empirically observed in failed software-intensive acquisitions, as in the acquisition of the Super Seasprite Helicopter (Ferguson & Blekin 2008).

Other factors have been reported as causes of failure in software projects. For example, lack of adoption of appropriate processes (DOD-US, 2000), unrealistic expectations (Standish Group 1994), inappropriate staffing and lack of management flexibility in being able to respond to changes on the way through the project (DAF 2000).

The US Department of Air Force (DAF 2000) reported that differences in the experience and competency of users, engineers and managers cause variations in the performance of the software development process. The lack of information and understanding of variation in the software development process is one of the causes of uncertainty that also contributes to failure of software-intensive projects.

Disciplined processes can reduce variation and uncertainty and improve the performance of software projects (GAO 2004; Humphrey 2002). However, the adoption

of such processes requires investment and long-term commitment, and they are often substituted for higher risk strategies that attempt managing uncertainty in the hope of obtaining short-term results.

Although disciplined processes are necessary to engineer products and systems, they may not be sufficient. As occurred in other sectors of the industry¹, to bring software development to an acceptable level of quality and performance will require the identification and adoption of a combined set of methods and strategies that, working as a system, would complement each other and lead to better results.

Disciplined processes can be complemented by organisational design supported by knowledge of variations and a better understanding of the software-intensive acquisition system, as suggested by Deming's System of Profound Knowledge² (Deming 2000). Pfleeger (1999) advocates the need for theories and models that fit the reality of software development to overcome the challenge of managing software projects. This is also in agreement with Deming's teachings.

Many aspects and findings of this research can also be directed to innovative engineering projects in general. The author chose to focus on the engineering of software-intensive systems not only because of his personal motivation and experience, but also due to the fact that the acquisition and development of software-intensive systems has a history of failure, difficulties and challenges that remain without answers.

It is in this spirit that this research proposes to investigate theories to improve our understanding of software-intensive acquisitions as systems and their variations, with the intent to find integrated strategies that would improve the acquisition and development of software-intensive systems.

¹ See for example the improvement in quality of automotive production by the Japanese industry achieved by lean production, which combines several quality improvements working as a system (Rechtin 2000 p. 28).

² Deming's System of Profound Knowledge identifies four parts to achieve effective management: (1) Appreciate the system; (2) Knowledge about variation; (3) Theory of Knowledge; and (4) Knowledge of psychology.

1.3. Engineering Software-intensive Systems

Software-intensive projects are engineering projects that rely on the development of software components to produce useful products. Therefore, the principles and practices of ‘good engineering’ should also apply to software-intensive projects. Software should be, as any other component, developed through engineering processes, but the history of failure of software-intensive projects and acquisitions suggests that is not true. Perhaps current software engineering processes are not yet adequate; or maybe there is something else beyond the engineering process that is needed to develop software effectively and efficiently.

Software is often a component of a large system with electronic hardware, mechanical parts, operational procedures, buildings, operators and users, amongst other components. Software solutions have a tendency to change more frequently than the hardware, because they are not necessarily constrained by the laws of physics until placed in the hardware that hosts them.

Since the software is responsible for the operation of the system, and often is the last component to be ready, it is also the one to be blamed for delays, cost overruns and the failure of the enterprise as a whole. Whether software is, or is not, the sole cause of troubled software-intensive projects, it is often perceived as the cause of failure.

Engineering of software-intensive systems is a relatively new field of engineering not yet mastered and maybe not even understood. Besides, software-intensive systems are usually unique and present problems never solved or seen before. For being non-repetitive in nature, the acquisition of software-intensive systems presents challenges for which there is no proven experience.

For these and other reasons not yet known, the acquisition, engineering and development of software-intensive systems deserve special attention, as will be shown below.

1.4. Significance of the Problem

The first open admission that there was a potential software crisis occurred in 1968 at the NATO Conference on Software Engineering in Garmisch, Germany (Dijkstra 1972). The statement suggested that it would not be easy to develop software programs with

the increasing power and sophistication of computers and the complexity of the problems to be solved.

Edsger W. Dijkstra, in his ACM Turing Award Lecture, ‘The Humble Programmer’, 1972, stated that:

(since 1968)... the machines have become several orders of magnitude more powerful! To put it quite bluntly: as long as there were no machines, programming was no problem at all; when we had a few weak computers, programming became a mild problem, and now we have gigantic computers, programming has become an equally gigantic problem. (Dijkstra 1972 p. 861)

As a consequence of Moore’s Law³ predictions, the fastest supercomputer in 2010 is 70 million times faster than the fastest computer in 1969⁴. Driven by the ever growing expectations of computer users, the sophistication and complexity of problems addressed by computer systems have increased several orders of magnitude. Today’s problems often call for solutions that require several computers working together connected by networks, often distributed over large geographic areas. To comply with safety requirements, back-up systems are required in the event of operational system failure so that lives and property are not put at risk.

The rate of change in the world, whether in the business, social or political environment, creates needs and problems that require sophisticated solutions that are often complex. These solutions must usually be found, developed and implemented within strict time constraints. Fuelled by the capacity and processing power of today’s computers and available technology, systems architects envisage solutions that push the boundary of software systems at an alarming rate.

To compound the situation, business creates a very competitive environment where customers seek the best deal and providers the highest profitability. In this context it is often the case that customers are lured by low price, vain promises and apparently

³ Moore’s Law is attributed to Gordon Moore who in 1965 predicted that the number of transistors per area in integrated circuits would double every two years (G.E. Moore, Electronics, Volume 38, Number 8, April 19, 1965).

⁴ In 1969 the CDC 7600 supercomputer could process 36 megaFLOPS (million FLoating point OPerations per Second) and in 2010 the Tianhe-IA supercomputer was capable of processing 2.56 petaFLOPS (quadrillion FLOPS) (source Wikipedia, <http://en.wikipedia.org/wiki/Supercomputer>).

favourable contractual conditions, and thus do not always choose the provider that would offer the best conditions to deliver a solution that would satisfy the customer's need. The US Department of Air Force Software Technology Support Center indicates that 'the competency of the contractor is the single most important ingredient in the recipe for successful contract performance' (DAF 2000 p. S-66).

For the software crisis we understand our inability to produce software products, or products that require software, on time, on budget and with the required operational quality. Forty years after the first signs of a software crisis and more than thirty years since Dijkstra's blunt realisation, software projects continue to be challenged in achieving established parameters of cost schedule and quality. Watts Humphrey, known as the 'Father of Software Quality' and one of the creators of the Capability Maturity Model (CMM) for software (SEI 2010), wrote in 2010 about the reasons why we continue having difficulties in managing large software projects and states: '... more and more large projects fail these days than in the past – and the failures are even more expensive and painful' (Humphrey 2010a p. 4).

1.4.1. Failure of Software Projects

The literature shows that software-intensive acquisitions and software projects are prone to failure (Flowers 1996; DOD-AU 2000; GAO 2004; McIntosh & Prescott 1999; Standish Group 1998). The history of defence procurement in Australia and abroad shows that success is the exception rather than the norm and frequently cost and performance have to be managed to face more important time constraints. Jones stated: 'large software projects are very hazardous business ventures' and 'for projects above 10,000 function points cancellations, delays and cost overruns have been the norm rather than the exception' (Jones 2006 p. 8).

The United States Department of Defence recognises the importance of software-intensive systems. The US DOD acknowledges the challenge of developing good quality software in face of the constant reduction of funds, increasing sophistication of defence systems and the systemic implications of managing software acquisitions (DAF 2000).

The Standish Group has surveyed and reported on the success and failure rates of Information Technology (IT) projects in the private and government sectors since 1994. The survey corresponding to the CHAOS Report (Standish 2003) indicates that 34% of

software projects in the United States are completed successfully, while 15% of software projects are cancelled before completion. The remaining 51% of software projects are considered to be challenged⁵. The percentage of projects that are cancelled or that do not achieve expected results corresponds to a waste of US\$55 billion of a total spending of \$255 billion in IT projects in the United States.

1.4.2. Failure of Software Projects in Australia

Software in defence equipment is pervasive and the majority of defence project delays in Australia can be attributed to problems with software development (DOD-AU 2000, 2001, 2002, 2003). Two of the main software-intensive acquisitions in Australia, the Collins Class submarine and the Super Seasprite helicopter programmes experienced schedule delays and compromised performance attributed to problems with the development of software (DOD-AU 2000, 2001, 2002, 2003; Hill 2002, 2003; McIntosh & Prescott 1999). The Collins class submarine had its combat system, the Tactical Data Handling System (TDHS) replaced after more than a decade of development (Yule & Wolner 2008). In March 2008 a decision was made to cancel the Super Seasprite helicopter programme after 12 years of development and seven years of delays (DOD-AU 2008).

The Australia Defence Materiel Organisation (DMO) indicates that Australian experience with schedule overruns in software projects is similar to the numbers reported in the Standish Group's CHAOS Report 1994, which shows the average of 222% in schedule overruns and that the organisation has been able to contain cost overruns through fixed-price contracts (McKinnie 2003).

The apparently favourable conditions of fixed-price contracts for an essentially unbounded product create the illusion of protecting the customer by shifting the costs to the contractor. Fixed-price contracts have been tried for 50 years without solving the problem of cost overruns (Humphrey 2010a). This approach however, may create a situation where both parties lose. McDowell stated that it is impossible to predict cost and schedule (McDowell 2008) and he attributes the delays to the RAAF's Wedgetail Airborne Early Warning and Control (AEW&C) aircraft to a fixed-price development program ‘where we always underestimate, which is why we (BAE Systems Australia)

⁵ Projects said to be ‘challenged’ deliver the end product to the customer but may experience delays, cost overruns and reduced functionality.

don't do them any more' (McDowell 2010). While cost overruns have been contained by fixed-price contracts, schedule overruns and reductions in performance can be seen in programmes like the Collins Class submarine (Yule & Wolner 2008) and the Super SeaSprite Helicopter (Ferguson & Blenkin 2008). Fixed-price contracts do not compensate for deficient specifications or contractors without the required experience to engineer and deliver the system, which results in the customer not getting what is needed. It can be argued that delays and the reduction of performance of expected capabilities can put at risk the ability of the Australian Defence Forces (ADF) to protect Australian territory.

1.5. Objectives

The objective of this research is to develop a better understanding of what drives the success or failure of software-intensive acquisitions and find ways to assist managers and decision makers to make the right decisions when facing the challenges of the complex environment of software-intensive acquisitions and projects.

1.6. Research Problem

The fundamental problem addressed by this research is formulated by the question:

Why is it so difficult to succeed in large software-intensive acquisitions and in what ways can the software development component of software-intensive acquisitions be adequately structured and managed to effectively achieve the established parameters of cost, schedule and quality?

The research adopts the premise that the effectiveness of software-intensive acquisitions in achieving the intended goal depends in large part on the behaviour of a social system, where people and organisations work together to achieve an intended common goal. Organisational structures are created by formal and informal relationships and processes. Intended goals are often expressed by established parameters of cost, schedule and quality, documented in contracts, specifications and plans. Quality is used here as a broad term that defines attributes and parameters of acceptance of services and products delivered by the contract.

The research follows the guidance provided by Deming's System of Profound Knowledge (Deming 2000). The System of Profound Knowledge offers a framework to transform and optimise management and provides the guidance to decompose the research problem into manageable sub-problems.

The System of Profound Knowledge, comprising of four interlocked parts, states that knowledge is built on theory and we must 'appreciate the system' and have 'knowledge about variations' in the system; and we must understand the 'theory of knowledge'.

To convey the essence of the theory of knowledge Deming explains that a system cannot understand itself and transformation requires a view from outside the system; and that it is management's job to direct all components of the system towards the intended common goal that is the aim of the system. According to Deming, management is about predicting the system's behaviour and without a theory, as simple as it may be, it is not possible to predict. Theories are built upon what is known about the system and should be continuously revised and extended as more is learned.

The fourth part tells us that managers need knowledge of psychology to be able to manage people. The System of Profound Knowledge will be presented in the review of the literature in Chapter 2.

Within these principles, the research will investigate the software-intensive acquisition as a system and its variations, and will formulate hypotheses regarding the development of a theory to improve prediction and instigate changes in management thinking to achieve better results.

The research problem is then decomposed into three sub-problems that address the three main aspects of Deming's System of Profound Knowledge: appreciate the system; understand variation; and develop hypotheses towards a theory of causes success or failure of software-intensive acquisitions.

1.6.1. Sub-Problem 1

Sub-problem 1 aims to investigate the software-intensive acquisition as a system and is formulated by the question:

What can be learned about the software-intensive acquisition system that would help to understand the system's ability in achieving its intended goal?

1.6.2. Sub-Problem 2

Sub-problem 2 aims to identify the causes of variation that ultimately drive the perceived variations of cost, schedule and quality and is formulated by the question:

What are the causes of variations in the software development component of software-intensive acquisitions and how do these variations affect the ability to achieve the established parameters of cost, schedule and quality?

1.6.3. Sub-Problem 3

The objective of sub-problem 3 is to capitalize the knowledge obtained from answering the questions posed by sub-problems 1 and 2, and propose a framework that would improve prediction and the chances of success of software-intensive acquisitions. Sub-problem 3 is formulated by the question:

How can a better understanding of the software-intensive acquisition system and its variations help to formulate a framework to improve prediction and the chances of success?

1.7. Research Approach

The research started by conducting a review of the literature in searching for information and theories that would help to answer the first two research sub-problems addressing two aspects of the System of Profound Knowledge: appreciate the software-intensive acquisition system; and understand the variations present in the system.

To be able to understand the social system of software-intensive acquisitions it is needed to understand its component, people and organisations, and how they interact. Also guided by the System of Profound Knowledge, the review of literature searched for answers in the fields of psychology and social sciences.

Following the literature review, the first sub-problem was addressed by an analysis of the software-intensive acquisition system aiming to develop hypotheses about the nature and behaviour of the system.

Similarly, the second sub-problem was addressed by an analysis of the variations in software-intensive acquisition system aiming to develop hypotheses about the nature of the variations in the system.

The hypotheses about the system and its variations were then tested for validity and combined into a framework that would improve prediction and ultimately success of software-intensive acquisitions, which is the objective of sub-problem 3. However, hypotheses that deal with social systems cannot be easily validated. A truly and incontestable validation should be obtained by testing the hypotheses on real case scenarios which is not always possible due to constraints imposed by cost, time and risk. For this reason, the research approach adopted modelling and computer simulation to test its hypotheses. A review of the literature in the fields of modelling and computer simulation was conducted to find the approach and techniques that would be best indicated for the task.

Adopting the chosen approach for modelling and simulation, a model of the software-intensive acquisition system was developed in accordance with the hypotheses proposed by the outcome of sub-problems 1 and 2. Using the model, pertinent scenarios were simulated to test the hypotheses that could be combined into a framework focussed on sub-problem 3.

In conclusion this thesis will present the findings of the research in response to the research problem and its sub-problems, together with its limitations and proposed future work.

1.8. Overview of this Thesis

Chapter 1 introduced and presented the overview of the thesis, motivation, the research topic and its questions, and the research approach.

Chapter 2 reviews the literature of pertinent topics to address sub-problems 1 and 2 including, reported causes of success and failure of software-intensive projects; systems and complexity theories; psychology and social theories.

Chapter 3 addresses sub-problem 1 and investigates the software-intensive acquisition system with the intention to develop hypotheses about how the system structure influences the system's ability to achieve the intended goals.

Chapter 4 addresses sub-problem 2 and investigates the causes of variation in software-intensive acquisitions, aiming to develop hypotheses about how variations in the system influence the system's ability to achieve intended parameters of cost, schedule and quality.

Chapter 5 reviews the literature of modelling and computer simulation, discusses the most appropriate approach and techniques for modelling the system, and proposes a model for a software-intensive acquisition system for testing the hypotheses formulated in Chapters 3 and 4.

Chapter 6 uses the model developed in Chapter 5 to simulate the software-intensive acquisition under conditions designed to validate the hypotheses formulated in Chapters 3 and 4.

Chapter 7 presents the findings of the research in response to the research problem, the limitations identified by the research and proposes future work.

The Appendices contain details of the modelling and simulation methodology, and articles published as result of this research.

Chapter 2: Literature Review

The literature review initially investigates the challenge to develop software, and the causes of success and failure of software-intensive acquisitions. Deming's System of Profound Knowledge defines the path for the research. In searching for a better understanding of the software-intensive acquisition system, its components and their variations, the literature review addresses the fields of systems, complexity, psychology and social behaviour applied to engineering and software-intensive projects. Engineering and software development processes are also part of the system and the review of the literature of these fields will be discussed in Chapter 3, dedicated to the software-intensive acquisition system.

The objective of the literature review is to aid the development of hypotheses that will help understand the root causes of the success and failure of software-intensive acquisitions and in this way help managers and decision makers to plan acquisitions that could have a better chance of success. Hypotheses will be tested in a virtual environment created by modelling and computer simulation. The review of literature of modelling and computer simulation will be presented in Chapter 5, and a model for the system proposed.

2.1. Success and Failure of Software-Intensive Projects

Many reasons have been offered to explain why software projects fail, yet software-intensive projects continue to present schedule delays, cost overruns and deliver products with reduced functionality, in many cases.

Dijkstra suggested that the problem is a consequence of the complexity of computers (Dijkstra 1972). The Standish Group reported several factors that contribute to the success of software projects (Standish Group 1995, 1999, 2001). Conversely, the absence of these factors would likely cause failure. According to the Standish Group the most important factors to succeed in software projects are: 'user involvement'; 'executive support'; 'clear statement of requirements'; 'clear business objectives'; and 'experienced project managers'. The list continues with less important factors such as: 'realistic expectations'; 'small milestones'; 'minimum scope'; 'formal methodology'; 'competent and hard-working staff'; 'reliable estimates'; and 'ownership'. Capers Jones

has also reported that inaccurate estimation and changing requirements cause software projects to fail (Jones 1995, 2006, 2008).

The list of factors that contribute to the success of software projects proposed by Boghossian (2002) is not very different. At the top of the list is ‘defined product development life cycle’, followed by ‘executive management support’; ‘user involvement’; ‘strong project management’; and ‘small and miniature project milestones’. The list continues with factors of less influence: ‘clear statement of requirements’; ‘realistic expectations on schedule’; ‘competent, trained and focused workforce’; ‘ownership’; ‘clear vision and objectives’; ‘software development and engineering practices (including well defined processes and software estimation)’; and ‘independent verification and validation’.

These factors, although insightful, do not identify the root causes of software project failure. It is important to understand and to know what would prevent ‘executive support’ and ‘user involvement’; or what causes the absence of ‘clear and stable requirements’; or why ‘small and miniature project milestones’ contribute to success. Surprisingly, and most likely erroneously, factors such as ‘competent and motivated staff’ and ‘software engineering practices’ feature as less important, suggesting that it is not so important how the software product is developed and built.

The history of failure of large software projects suggests that the root causes lie well beyond the complexity of computers. It extends to the complexity of the systems we want to build that reflect the high expectations we place in what computer systems can do (more, better and faster) and how we should be able to do it (quicker, cheaper, more accurately and more precise). The problem unfolds when reality does not match the expectations.

2.1.1. Success of Software-Intensive Acquisitions

The success of software-intensive acquisitions depends on the success of software development projects,

Managing software-intensive acquisitions is not an easy task and management has been identified as one of the main causes of failure (DAF 2000; Flowers 1996; GAO 2004). Software-intensive acquisitions require the cooperation of multiple organisations and present no central management. In practice, a network of distributed management

manages the system through contracts, specifications, schedules and budgets that flow amongst organisations, from the customer to the prime-contractor and then to subcontractors. Status reports and contractual deliverables flow back from contractors to the customer organisation. However, contracts and specifications are not perfect and free of conflicting interpretations; schedules and budgets are not always correct and reports reflect only part of the reality. In practice, each organisation will manage their contracts to meet their own interests. What is best for individual components of a system does not guarantee the best result for the system as a whole (Deming 2000).

The history of delays, cost overruns, reduced performance and total failure of software-intensive acquisitions shows that managers have not performed well (ANAO 2008, 2010b; DAF 2000; DOD-AU 2000, 2001, 2002, 2003; GAO 2004; Hill 2002, 2003; McKinnie 2003). When facing complex problems, the capacity of decision making is limited by human cognition; it can be based on flawed mental models, intuition or inappropriate heuristics; it is influenced by the impact of the decision on the organisation and on the individual and can get things wrong (McLucas 2001).

2.1.2. Cobb's Paradox

The failure in identifying the root causes of success and failure of software projects is expressed by Cobb's paradox (Martin Cobb⁶ in Standish Group 1995): 'We know why (software) projects fail; we know how to prevent their failure – so why do they still fail?'

In response to Cobb's paradox, Kasser stated: 'a paradox is a symptom of a flaw in the underlying paradigm' (Kasser 1998 p.9).

Cobb's paradox and the likely flaw stated by Kasser, provide the motivation for this research. Knowing some of the perceived factors that seem to contribute to the success or failure of software projects, most likely will not address the root causes of success and failure of large software-intensive projects. Therefore, we must continue searching for paradigms that would better explain the reality of large software-intensive projects and software-intensive acquisitions.

⁶ Martin Cobb, Treasury Board of Canada Secretariat, Standish Group, 1996.

2.1.3. The First Clue

Problem solving and software development are creative tasks that depend on communication between people (gathering of information, expressing needs and possible solutions) and cannot be automated (DeMarco & Lister 1999). The development of software is therefore strongly dependent on people's performance and behaviour. People are the most important resource in the development of software (Brooks 1995), and perhaps the highest risk. Turner and Boehm explain that:

In essence, software engineering is done 'of the people, by the people and for the people', because 'people organize themselves into teams to develop mutually satisfactory software systems'; 'people identify what software capabilities they need, and other people develop these for them'; and 'people pay the bills for software development and use the resulting products (Turner & Boehm 2003 p. 4).

The reasons for success or failure of software-intensive acquisitions are better expressed by Stephen Flowers:

The success or failure of a major software-intensive acquisition program depends on a highly complex combination of factors, not all of which are under the control of the acquisition manager... Critical failure/success factors are the crucial risk elements of a software acquisition program. They may be managerial, financial, technical, human, or political in nature. Program success or failure occurs as a result of both subtle and obvious interactions among all factors. When one or more factors are in a less than optimal state, they increase the likelihood that a program will fail, or worst case — become a disaster. (Flowers in DAF 2000 p. 2–15)

The success of large software-intensive acquisitions depends not only on how the software is developed but also on the behaviour of a diverse and complex social system that comprises the software-intensive acquisition, where people and organisations interact and cooperate to specify and develop products highly dependent on the software. All organisations in the acquisition contribute to the overall behaviour of the system and management, and in principle, have the responsibility to make the system operate effectively and efficiently.

Flowers, Brooks, Boehm, DeMarco and Lister provided the first clue to reveal what is behind ‘Cobb’s Paradox’ and the root causes of success or failure of software-intensive projects. The clue points to the complexity of a system where people are the main components. Complexity arises in the way the multiple factors influence each other and the overall behaviour of the system, and people influence these factors. Understanding these many factors, their interdependency and how they impact on the system’s outcomes, is the objective of this research.

2.1.4. The Importance of Knowledge

The need for knowledgeable, talented and motivated people to develop software products is undeniable (DAF 2000; SEI 2001). Knowledge is not only of fundamental importance to providers that formulate the problem and engineer the solution, but also to customers in their ability to communicate the need (DAF 2000). Lack of knowledge management also has been reported as one of the causes of failure of software projects (Perkins 2006).

In response to deficiencies in knowledge, the GAO report on ‘Defence Acquisitions, Assessment of Selected Weapon Programs’ (GAO 2009) recommends a knowledge-based acquisition approach to address the pervasive problems of cost and schedule variation. The ISO/IEC standards on systems and software engineering processes (ISO/IEC-12207; ISO/IEC-15288 2008) have included processes to provide a supply of skilled and experienced personnel that are required to achieve organisation, project and customer objectives. The activities and tasks of these processes include the identification of skills required for the project and the skills available; the development and acquisition of skills in case there is a gap between those required and those available; and knowledge management.

Unrealistic Expectations

Stakeholders in traditional project management often assume that foreknowledge is perfect and that:

We can define complete, consistent, testable, and buildable requirements; decompose perfect requirements to perfect specifications; accurately estimate effort, cost, and schedule for the specifications; schedule work according to

this information early in the program; and measure progress using earned-value management or similar techniques. (Turner 2007 p. 12)

More often than not, this assumption becomes an unrealistic expectation. Turner and Boehm report that ‘most software people do not do well at expectation management’; and that ‘the differences between successful and troubled software projects are most often the difference between good and bad expectation management’ (Turner & Boehm 2003 p. 7).

Turner and Boehm continue saying that: ‘(software people) have little confidence in their ability to predict software schedules and budgets, making them a pushover for aggressive customers and managers trying to get more software for less time and money’ (Turner & Boehm 2003 p. 7).

Software-intensive products are an encapsulation of knowledge into software. The activity of developing software involves acquiring specific types of knowledge to solve the problem, referred to as domain knowledge, and then apply knowledge of software technology to translate the solution into the code (Armour 2004 p. 7). Managers often consider the latter and forget that only after the problem is solved can the software code be completed.

It is important to observe that the process of acquiring knowledge takes time and effort and it can be equated to cost and schedule. Lack of knowledge, experience and skills is thus another factor that causes projects to under-perform.

2.1.5. The Second Clue

The importance of knowledge in software-intensive acquisitions and the impact that the absence of knowledge and skills cause to the project, provide a second clue to find the root causes of success or failure of software-intensive projects and reveal what is behind ‘Cobb’s Paradox’.

Software-intensive projects and acquisitions may fail because of the lack of technical knowledge to engineer the solution, and the lack of management knowledge to recognise and plan for this deficiency. This assertion is supported by GAO and ISO/IEC (GAO 2009; ISO/IEC-12207 2008; ISO/IEC-15288 2008), who recommend a knowledge-based acquisition approach executed by skilled and experienced personnel, to address problems of cost and schedule variation and achieve the intended objectives,

as is to be discussed in Section 2.2.4. Therefore, it is reasonable to hypothesise that when projects are not performing as expected and interests are at risk, decisions are made within constraints that are unlikely to address the lack of knowledge. These constraints often produce undesirable social behaviours that decrease motivation, discourage learning and cooperation and worsen the situation.

Understanding how knowledge or lack thereof impact the system's outcomes, and what structures will favour the acquisition and dissemination of knowledge, is the objective of this research.

2.2. The System of Profound Knowledge

Deming's System of Profound Knowledge (Deming 2000) was chosen to guide this research. The System of Profound Knowledge is intended to aid the transformation of management style to improve the performance of business enterprises, and it fits well to address the research problem and as an approach to investigate the first clue identified in the previous paragraphs.

W. Edwards Deming⁷ was instrumental in the way that Japan built a reputation in high-quality products and innovation through the application of quality management and control in manufacturing and services industries, later also introduced in the USA. The success of Deming's methods is attributed to the way that quality management and control is implemented as a system where individuals and organisations are responsible for quality and quality improvements. Later in life Deming realised that Total Quality Management (TQM) had become a superficial label for tools and techniques that could have different interpretations, often driven by short-term performance improvements. The real work suggested by Deming was in the 'transformation of prevailing management style' (Senge 2006 p. xii).

Deming claims that the principles that drive the prevailing management style of organisations in industry, government and education are inappropriate and in fact inadequate to achieve effective management and a transformation is needed.

⁷ Dr William Edwards Deming (1900–1993), American statistician and consultant in the fields of quality control and management.

The transformation, he claims, starts with the individual, who by understanding the principles of the System of Profound Knowledge will be able to carry out the transformation of the team, the division, the organisation and the system as a whole.

Deming's System of Profound Knowledge is defined in four parts:

- 1) Appreciate the system
- 2) Knowledge about variation
- 3) Theory of knowledge, and
- 4) Knowledge of psychology.

Deming stresses that these four parts are interdependent and will not produce the intended results if applied in isolation. They are part of a system: the System of Profound Knowledge.

2.2.1. Appreciate the System

The first part of the System of Profound Knowledge is grounded on the concept of 'system' as 'a network of interdependent components that work together to try to accomplish the aim of the system' (Deming 2000 p. 50). There are several implications behind this concept when applied to the joint business enterprise of software-intensive acquisitions, as discussed below.

Deming explains that every system must have an aim. Without an aim there is no system. The question is: what is the aim of the system? In joint business enterprises, the aim of the system is the 'common goal' stated in contracts, specifications and other documents that are often incomplete and leave a margin for interpretation and do not clearly and unquestionably convey the 'common goal'.

A system is a network of independent components. In joint business enterprises people and organisations are the independent components that have their own abilities, deficiencies, strengths, weaknesses, interests and motivations. In reality, each component in the system behaves in accordance with an individual and intrinsic characteristic and has an individual interpretation of what is the 'common goal'. It is not surprising the concept of system expressed by Deming states that the components of the system work together 'to try' to accomplish the aim of the system. Ultimately, people

and organisations try to accomplish a common goal, without a certainty that all are aiming for the same target. Even when the best of intentions are present, there is no guarantee that the effort applied in the collective task will be adequate to satisfy the many perceptions and interpretations of the ‘common goal’. For this reason the system must be managed as the system is unable to manage itself.

All systems have an aim, but the aim of the system does not always coincide with the ‘intended aim’ and the system may not be capable of achieving the ‘common goal’. The system must be managed to align the ‘intended aim’ with the aim of the system. Understanding the system is an imperative to manage the system to achieve what is wanted from it.

Deming explains that the components of the system should work together so that the system as a whole achieves the intended purpose. Each component should contribute to the effectiveness of the system and sometimes, individual components may have to carry inefficiencies to contribute to the common good. When all components individually aim to achieve maximum efficiency, the system may not achieve maximum effectiveness.

‘Appreciate the system’ aligns with the need to understand software-intensive acquisitions as systems to be able to find the root causes of success and failure. Systems and complexity are relevant topics to include in this literature review and will be addressed further in this chapter (see Section 2.3). The software-intensive acquisition system will be discussed in Chapter 3 with the intent to identify the system’s components and interactions and how these contribute to achieving the intended aim of the system.

2.2.2. Knowledge about Variation

Variation is the spread of the values of an attribute. When associated with an attribute of an entity or object, variation gives an indication of the range of values the attribute can assume, and the way these values are distributed around the mean or typical value. The term variation is also used to express the difference between what is expected and what is observed. Numerically, variation is the difference between an expected (or planned) value and the actual value.

Systems are subject to variation. Variations present in input parameters, in the internal components of the system and in the environment, influence the system and what the system produces as output. Variations are not all bad; some are very good. In fact ‘life is variation’ (Deming 2000 p. 98). Variation is the source of variety⁸; it is what makes people and things different. Variation is present in any process. Variation exists in industrial and manufacturing processes, in engineering processes and in software development.

Deming explains that variations occur due to common causes or special causes. Variations due to common causes are associated with fluctuations that are characteristic of components in the system; are within a limited range; have a mean value and a probability of distribution, and thus can be predicted. Variations due to special causes are created by events that are not part of the system of common causes and cannot be predicted. Deming claims that most problems with services and production lines are caused by common causes and warns that two mistakes are frequently made:

(1) to react to an outcome as if it came from a special cause, when actually it came from common causes of variation; (2) to treat to an outcome as if it came from common causes of variation, when actually it came from a special cause. (Deming 2000 p. 174)

To be able to predict the variation in the system we must know the common causes of variation and how these impact the system, and eliminate special causes of variation or else make the system resilient to them.

As an illustration, variation in manufacturing processes is caused by variation in the input materials, in the machines that transform inputs into output products, in the environment and in people that operate the process. These are common causes of variation, and knowledge about them allows prediction of the variation in the output products of the system. By specifying margins of tolerance of variation in input products and in the transformation process, the output product can be maintained within specified parameters of quality.

⁸ See the concept of ‘variety’ from Cybernetics in Section 2.3.1 further in this chapter.

Stable Processes

When the mean value and the probability distribution of variation are known, the system is classified as stable. Under these conditions the system can be placed under statistical control, allowing prediction of future behaviour. Prediction will have a component of uncertainty if some of the common or special causes of variation are unknown. Common causes of variation can be identified and understood by knowing the system, while special causes of variation may not.

Industrial and manufacturing processes fall in the category of stable processes. The variations in the inputs are known, the process is well understood and outputs can be predicted to be within known limits of variation. These kinds of processes can be partially or fully automated and placed under statistical control. Human intervention is often kept to a minimum. If at any point of time the output of the process exceeds the expected variation it is because the process is not completely understood⁹ or was caused by a special cause of variation¹⁰. In the first case, the process can be modified to compensate for the newly discovered common cause of variation; in the second case the event that caused the variation should be prevented or the process made resilient to its occurrence.

Creative human-intensive processes like engineering and software development are more difficult to place under statistical control and achieve the stability required, because the causes of variation may not be well understood. Variations in cost, schedule and quality are typically consequences of other, and more fundamental, causes of variation.

Variability and Uncertainty

Variability and uncertainty reduce our ability to predict the system behaviour (Vose 2000). Vose explains that variability is a characteristic of the system driven by chance, while uncertainty is originated by lack of knowledge about the system. Uncertainty, according to Vose, is subjective and can be reduced by further study, measurement or anything that can improve our knowledge about the system, though it may not be

⁹ It could be discovered that the variation of the input is greater than initially thought and the process has to be modified; or the operator of the process was not trained to deal with common cause of variation and the operator needs further training.

¹⁰ It could have been a water leak from a fire sprinkler that damaged one of the sensors of the machine.

eliminated, while variability cannot be reduced by additional information and knowledge, and may be reduced only by changing the system.

2.2.3. The Third Clue

Variability and uncertainty provide the third clue to explain Cobb's Paradox and reveal the flaws in the current paradigm for software development and software-intensive acquisitions.

In software development variation is associated with the production rate and quality of software (SLOC/person.day or FP/person.day; and defects/ KSLLOC, defects/Pages of Documents). However, these are consequences caused by other forms of variation due to variations in people's performance. Variations in people's knowledge, skills, experience, behavioural style, and motivation are likely to be the drivers that cause variation in software products. Uncertainty created by lack of knowledge about variations in the system makes almost impossible the estimation, planning and ultimately the predictions of future behaviour and outcomes, of the software-intensive acquisition.

'Knowledge about variation' aligns with the need to reduce uncertainty and understand the root causes of success and failure of software-intensive acquisitions. Variation in software-intensive acquisitions will be discussed in Chapter 4 with the intent to identify common causes of variation and how these impact the perceived variation of cost, schedule and quality.

2.2.4. Knowledge of Psychology

Deming suggests that knowledge of psychology helps to manage people effectively. Psychology in the context of this research is what scientists and philosophers of various persuasions have created to try to fulfill the need to understand the minds and behaviours of human beings as individuals¹¹. Psychology provides the foundation of a theory of human behaviour that would allow prediction of what a person might do under certain circumstances, and most importantly, what would motivate people to do what is expected of them. Knowledge of psychology helps to understand what motivates

¹¹ Definition adapted from the Dictionary of Psychology (Reber & Reber 2001)

people, and what turns them away from performing their tasks well. Undoubtedly managers could make good use of this knowledge.

Understanding people as individuals helps to understand a social phenomenon but it is not sufficient. People integrated in social systems like teams and organisations will not behave as individuals in isolation because they are influenced by others and by the social environment. Therefore, psychology should be complemented by the knowledge of group behaviour. Psychology and group behaviour are relevant topics to include in this literature review and will be addressed further in this chapter (see Section 2.4).

2.2.5. Theory of Knowledge

According to Deming, experience without theory has no practical use because without theory it is not possible to predict. Information is obtained by analysis and understanding the data collected through experience or experiments. Knowledge is acquired by further understanding information. However, knowledge does not yet provide ways to predict future behaviour and for that, theories need to be developed.

Deming asserts that a theory can be simple and may not be complete or perfect and still be useful, so long as it provides a useful basis for prediction (Deming 2000 pp. 102–103). If in practice the prediction is proved to be incorrect, the theory should be reviewed and modified.

Experience is based on observation and may help an individual to make choices and decisions. Personal experience, however, is not always sufficient for effective prediction and most likely is not useful to others that did not have the same experience. Experience placed into the context of a theory can provide ways for effective prediction not only for the one who experienced but also others. Like models, theories may be useful and can be expanded, reviewed and changed to continue being useful under new and broader conditions.

This research will formulate hypotheses about the software-intensive acquisition system and its variations. Hypotheses will be tested in a computer-simulated environment and will aid developing a framework for software-intensive acquisitions, being a set of assumptions, concepts, values, and practices that constitutes a way of viewing the reality of software-intensive acquisitions. After being validated in practice it is expected that the framework would lead to a theory of software-intensive acquisitions.

2.3. Systems, System Thinking and Complex Adaptive Systems

The concept of a system for the purpose of this research was introduced early in this section (see 2.2.1) as adopted by Deming in the System of Profound Knowledge, repeated here.

For the purpose of this research the concept of system is a variation of Deming's definition introduced early in this chapter (see 2.2.1), with the inclusion of the word '**intended**':

*System is a network of interdependent components that work together to try to accomplish the **intended** aim of the system* (based on Deming 1994 p. 50).

Most likely the definition above is what Deming had in mind to refer to man-made systems intended to achieve an explicit aim. The inclusion of the word 'intended' is to make clear that man-made systems have two aims: the system's implicit aim and the aim that is intended for the system. The implicit aim is what the system is capable of doing and it does not always coincide with what is intended and expected of the system. The 'purpose of the system' will be discussed further in this section.

The term 'complexity' is not easily defined. There is little consensus on a precise definition of 'complexity' (Axelrod & Cohen 2000 p. 16), and often the concept of 'complexity' is expressed by examples within contexts of interest. However, Axelrod is sympathetic to the proposal of Gell-Mann (Gell-Mann 1995) suggesting that: 'a system should be called complex when it is hard to predict not because it is random but rather because the regularities it does have cannot be briefly described' (Axelrod & Cohen 2000 p. 16). The increase in diversity of interconnected and interdependent components of a system drives the growth of complexity (Kauffman 1995 pp. 88–87, p. 296).

This research would not benefit from a precise mathematical definition of complexity. Instead, and without attempting to define complexity, this research accepts that complexity expresses the act of being complex, and the degree of difficulty in explaining and understanding a system because of many interacting and interdependent parts, each with an individual function contributing to the behaviour of the system.

Software-intensive products are complex technical systems with many interconnected components (software, hardware and documentation), engineered by an also complex

social system, comprised of people associated in teams and organisations, bounded by relationships, processes and contracts.

Technology alone fits no purpose. The purpose of technology is to bring some form of social benefit, whether to one or many individuals. What follows is an extract from reflections upon the philosophy of socio-technical systems (Ropohl 1999). The concept of socio-technical systems became a construct to explain technology generally. The realization of socio-technical systems follows the basic model of technical development: a social process where science and technology are necessary but not sufficient conditions, and a socio-technical system itself. The performance of socio-technical systems is determined by a combined interaction of technology and social factors, the latter also encompassing economy.

Software-intensive systems, as technological artefacts used and operated by people, are socio-technical systems; and software-intensive projects and acquisitions are also socio-technical systems themselves where people make use of technology to develop software-intensive systems. Performance is therefore determined by a combination of technology and social factors.

2.3.1. Systems Theory and Cybernetics

Systems theory began in the late 1940s with the work of Wiener on cybernetics (Wiener 1948) followed by Ashby's 'Law of Requisite Variety' (Ashby 1957), an important contribution to Cybernetics. Bertalanffy (1968) proposed the General Systems Theory, as yet another attempt to unify a theory that would be applicable to any kind of systems.

Systems theory is an inter-disciplinary field and applies to systems of different kind and levels of complexity, whether the system may be natural, man-made, biological, mechanical, technological or social. Systems theory attempts to explain, control and predict the behaviour of systems. Systems theory moves from reductionism to holism. The first suggests that the whole can be understood by decomposing it to its parts, while the second adopts the premise that the whole can be understood only when all its parts are working together. The holistic approach conforms to the system definition, as components in isolation do not constitute a system. A system is more than the sum of its parts.

Analytical tools provided by systems theory can be helpful for quantitative analysis of simpler systems; however they are yet inadequate when applied to more complex systems, in particular social systems. Concepts introduced by systems theory and cybernetics can be powerful tools for qualitative analysis of complex systems. Given the nature of software-intensive acquisition systems, this review will focus on ‘systems concepts’ and not on ‘systems analytical tools’.

Cybernetics

Cybernetics, as defined by Wiener, is: ‘the science of control and communication, in the animal and the machine’ (Wiener in Ashby 1957 p. 1). Ashby defines cybernetics as: ‘the art of steermanship’ (Ashby 1957 p. 1).

Variety

One of the central aspects of cybernetics is the concept of variety. Variety expresses the total number of possible states of a system, or of an element of a system (Beer 1972). The state of the system is the collective state, or value, of each attribute of the system at one point of time. The Law of Requisite Variety (Ashby 1957) states that for effective control, the controller must have at least the same variety as the entity, or system, being controlled. Only variety can absorb variety (Beer 2002). Requisite variety is the kind of variety that would bring the system into the desired state.

Stafford Beer applied cybernetic principles, and in particular the Law of Requisite Variety, to organisations and management and developed the concept of the ‘Viable System’ (Beer 1979). The aim of the Viable System is realizing the Law of Requisite Variety: output variety is equal to or greater than the input variety. When applied to organisations, it means that the organisation should provide the variety required to deal with the variety that enters the system to produce the desired variety that leaves the system.

Stafford Beer states that variety is a measure of complexity (Beer 1979). To quantify variety, it would require a complete count of the number of elements in the system and how these elements are inter-related and influence each other, which would lead to estimating the number of possible states in the system. Although quantifying variety can be difficult, by attempting to do so, even if not complete, it is possible to obtain a qualitative estimation of the complexity of the system and how to control it.

In the context of software-intensive acquisitions, variety is present in the socio-technical system. It is expressed by the many states the system can attain in accordance with the task the project aims to accomplish, and the resources needed to perform the task within the established performance of performance of cost, schedule and quality. The variety of the technical system represents the number of components in the system needed to describe the customer's need; to define the problem to be solved and specify the solution that when implemented, should satisfy the need; to define and implement the solution through specification and design of hardware, software and operational procedures, as well as in the process and procedures to develop and verify the solution.

The variety required to absorb the variety of the technical system should be present in the social system. The social system should be able to control the technical system, as well as controlling itself. Domain experts, operators, engineers and technicians provide the variety to transform a perceived need into explicit need. The explicit need is transformed into a statement of the problem and this into a system specification. The system specification is transformed into a system design and then into the specification and design of hardware, software and operational procedures, which in turn will be transformed into operational procedures, hardware and software products as well as test procedures to verify the effectiveness of the solution. Variety is present in people's personality, knowledge, experience, expertise, creativity and their capacity to communicate and work together.

Disobeying the Law of Requisite Variety, according to Stafford Beer, is one of the greatest problems that cause the failure of control systems, including the control of the firm through leadership and management:

For management always hopes to devise systems that are simple and cheap, but often ends up by spending vast sums of money to inject requisite variety – which should have been designed into the system in the first place. (Beer 1972 p. 54)

Variety can be associated with the many factors that contribute to the success or failure of software-intensive acquisitions suggested by Flowers (1996) (see Section 2.1.1). Although, not all aspects of variety are under the control of managers, managers are expected to create the conditions to enable the required variety to bring the system to

desirable states, which can become extremely difficult when the system is not prepared to achieve its intended purpose.

The Law of Requisite Variety shows that to be effective, a system has to be at least as complex as the task it intends to execute (Bar-Yam 2003), implying that technical systems have to be as complex as the need they should satisfy, and the social systems have to be as complex as the technical systems they are tasked to engineer.

Diversity that is useful to satisfy the Law of Requisite Variety (Ashby 1957) increases the robustness of social and technical systems, allowing them to achieve their goals under adverse and unplanned conditions (Axelrod & Cohen 2000; Hitchins 1992). Variety in technical systems is present in the form of features that may be useful for unknown situations, or a spare computational capacity to expand the system's capabilities that might be required in the future. In social systems diversity comes in the form of human resources that provide knowledge, expertise and ideas.

The Purpose of the System

Stafford Beer stated that every system has a purpose and 'the purpose of a system is what it does' (Beer 2002 p. 218), which becomes a powerful concept if not a principle. The implication this insight brings is that if 'what the system does', does not reflect the intended purpose for the system, most likely the system will have to be changed. For organisational systems that have an enterprise as a goal, Beer explains that changing the system is about changing the structure that makes the system do what it is required to do and changing the system is a responsibility of management. Beer's ideas about the purpose of the system and how management can bring the purpose of the system into alignment with the intended purposes conforms to principles embedded in the System of Profound Knowledge.

Engineering and software-intensive projects are social systems structured hierarchically in accordance with roles and responsibilities that require specific levels of knowledge, skills and experience. Cost, schedule and quality are the three main parameters used to establish the goal and to evaluate how well the system is achieving the intended goal. Quality is expressed by sets of specification, requirements and measures of performance of the product the system shall produce. Cost and schedule are the enablers of resources that will be applied to achieve the intended quality of the product.

The challenge presented to management is about establishing what is wanted from the system (cost, schedule and quality), estimating the capacity of the system (human and other resources) in achieving the intended goal and making changes to the goals as well as to the system itself (resources and structure), when reality does not conform to what was planned. Changing goals and the system may cause conflicting responses that become management dilemmas. Changing requirements and measures of performance quite often increase costs and extend schedules, hence management resistance to these changes. Without changes in requirements the product may not be able to perform the way it should to solve the customer's need.

It is important to observe that engineering staff, customers and management are all components of the system and contribute to the capacity of the system in achieving its goals, whether intended or required. Management as well as customers should be able to understand and evaluate the changes and their consequences if these are necessary to make the system achieve its realistic goals.

Realistic cost and schedule estimation should take into consideration both the technical and the social systems. However, the complexity of the systems and the unprecedented task make it extremely difficult to obtain accurate data for realistic estimation. In addition, business and political factors often take precedence for estimating cost and schedule, leading to unrealistic goals.

If, for example, the engineering resources available in the system do not provide the requisite variety, i.e. in this case skills and attitudes to engineer the solution, the deficit of variety will have to be acquired, which will take time and expense. If the requisite variety is not available distortions occur, which in turn reduce quality. To bring quality to acceptable levels will incur cost and take time. The management challenge is then to envisage ways to provide requisite variety without increasing cost and schedule, which is not always possible.

Effectiveness, Efficiency and Performance

Effectiveness indicates how well a system achieves its goals. The effectiveness of a technical system is determined by its capacity to execute the task required to satisfy the need. The effectiveness of the social system represents its capacity to engineer the technical system.

Efficiency is an indication of how much waste the system produces to be effective. The efficiency of social systems that engineer technical systems can be evaluated based on the ratio of effort required to execute ‘end tasks’ and ‘enabling tasks’, associated respectively with ‘end-products’ and ‘enabling-products’ (EIA 1999)

To assess the effectiveness and efficiency of a real system, it should be compared with the ideal system that is 100% effective (Hitchins 1992). The ideal technical system satisfies the need and does not waste energy or computational resources. The ideal social system does not waste human and other resources. In the ideal social system every person knows what is needed to engineer the ideal solution, and no effort is spent on ‘enabling-tasks’, rework or learning.

Each system has a nominal efficiency to achieve maximum effectiveness (Hitchins 2003 p. 56). A system can be highly efficient and not effective. Considering the number of problems fixed by unit of time, a software development team can be very efficient by addressing only the software problems that are easy to fix. If the critical and difficult problems are not resolved, however, the technical system may not be operational and the software development team would not be effective in achieving its ultimate goal.

Performance is an indication of how well the system does what it is supposed to do, and conveys the level of ‘satisfaction’ or ‘value for money’ that is therefore subjective. There is no acceptable definition of performance in ways that it can be applied to different processes (Aslaksen 1996). This research needs a definition of performance that can be used to compare systems of the same kind that perform a process of finite duration and produce an output product, taking into account effectiveness, cost, duration and the efficiency of the process. First the performance of the ideal system will be calculated and then used to compare against the performance of the real system. The method to calculate and compare performance will be presented in Chapter 3.

2.3.2. Systems Thinking

‘Systems thinking’ is an approach to solve problems within the context of their surroundings and not in isolation. Systems thinking adopts a holistic approach instead of

reductionism¹². Holism when applied to systems is a paradigm concerned with wholes and their properties (Checkland 1993 p. 13). The holistic paradigm supports the notion that investigation and analysis of a system cannot be reduced to the analyses of its components without considering how they are connected and interrelated. Reductionism is an approach to scientific investigation and analysis of complex problems (and systems) by breaking them into their more treatable components.

Checkland defines systems thinking as:

an epistemology¹³ which, when applied to human activity is based upon four ideas: emergence, hierarchy, communication and control as characteristics of the system. (Checkland 1993 p. 318)

Within the context of this definition Checkland defines emergence as,

the principle that whole entities exhibit properties which are meaningful only when attributed to the whole'; hierarchy as, 'the principle according to which entities meaningfully treated as wholes are built up of smaller entities which are themselves wholes ... and so on'; communication as, 'the transfer of information'; and control as, 'the process by means of which a whole entity retains its identity and/or performance under changing circumstances.

(Checkland 1993 pp. 313–314)

Applying a systems thinking approach to management Senge (2006) proposes five disciplines for developing three learning capabilities that should build a learning organisation (see Table 1). Senge advocates that only organisations that are capable of learning are able to face the challenges of a complex and continuously changing environment, conforming to the theory of natural selection (Darwin 1859), commonly known as ‘survival of the fittest’¹⁴, because Darwin with ‘fittest’ meant ‘the ability to

¹² Reductionism is an approach to scientific investigation and analysis of complex problems (and systems) by breaking them into their more treatable components. Reductionism can be seen as the opposite of holism.

¹³ Epistemology is a theory concerning the means by which we may have and express knowledge of the world (Checkland 1993 p. 314).

¹⁴ The term ‘survival of the fittest’ was coined by Herbert Spencer after reading Charles Darwin’s *On the Origin of Species* (Darwin 1859) (source Wikipedia http://en.wikipedia.org/wiki/Survival_of_the_fittest).

fit’, that is the ability to learn and to adapt. ‘Systems thinking’ is the ‘fifth discipline’ that fosters the understanding of complexity.

Table 1 – Senge’s Five Disciplines

	Discipline	Capability
I	Personal Mastery	Fostering Aspiration
II	Shared Vision	
III	Mental Models	Developing Reflective Conversation
IV	Dialogue/Team learning	
V	Systems Thinking	Understanding Complexity

Influenced by Deming, Senge understood that the ‘transformation of the prevailing system of management’ goes beyond tools and techniques, often adopted by managers seeking localised improvements and short-term results, and for a lasting transformation to occur there is required a ‘profound knowledge’. Senge discusses the elements of Deming’s ‘system of profound knowledge’ and maps them to the five learning disciplines, as shown in Table 2.

Table 2 – Five Disciplines mapped to Deming’s System of Profound Knowledge

System of Profound Knowledge	Five Learning Disciplines
Appreciate the System	Understanding Complexity through System Thinking
Theory of Knowledge	The importance of Mental Models
Knowledge of Psychology (specially ‘intrinsic motivations’)	Personal Vision Genuine Aspiration
Knowledge about Variation	Not mapped

Senge does not map ‘knowledge about variation’ to his five learning disciplines, as he indicates this aspect of the ‘system of profound knowledge’ is associated with statistical theory and method. However, knowledge about variation is not only about statistical control, it is about understanding how variation occurs, its causes, possible control and likely consequences, well addressed by Deming and discussed earlier in this chapter.

2.3.3. Complex Systems

Complexity theory, developed in the 1990s by several scientists mostly from the Santa Fe Institute in the USA, extends systems theory to systems that were previously considered untreatable, including chaotic and complex systems.

The normal usage of the term chaos implies randomness and disorder. A truly chaotic system is not deterministic and therefore unpredictable. Systems that present high-dimensional chaos behave in accordance with a large set of rules and display little structure. These systems are closer to the normal usage of the term chaos and thus are also unpredictable. Systems with low-dimensional chaos¹⁵ have a global structure and can be described by a small set of rules. Prediction of future behaviour may be possible for the short-term but not for the long-term. This kind of system can be described by a few mathematical rules but the presence of instabilities makes the system very sensitive to initial conditions, and long-term prediction extremely difficult. The weather is a typical example of low-dimension chaotic system and in essence a complex system.

Complex adaptive systems are open systems with components interacting with other components and with the environment and by doing so they change themselves, causing changes in the system. This form of change is called ‘adaptation’. In this category are included social systems and social organisations, which have special interest for this research. Complex adaptive systems will be discussed further in this chapter.

Emergence

Emergence is the central property of systems and represents what the system does that its components in isolation are not capable of doing. Emergent properties are the aim of the system, and the intent of systems and complexity theories is to provide ways to predict and even design systems with specific and desirable emergent properties.

Holland (1995) reflects upon a simple manufacturing production line that achieves a collective productivity greater than the sum of individual workers, observing that there are no models to demonstrate how the transition between individual to collective production occurs and raises questions about what are the adaptive mechanisms that

¹⁵ Low-dimensional chaos has a precise mathematical meaning: behaviour that has global structure; can be described by few mathematical rules; sensitive to initial conditions due to instabilities determined by the presence of positive Lyapunov coefficient (Stacey 1999).

favour the emergence of desirable and persistent properties. Although the creative process required to develop software is more complex than a repetitive manufacturing production line, it is reasonable to assume that similar questions also apply to software-intensive acquisitions.

What are the desired emergent properties of the software-intensive acquisition system? What actions of individual actors, and the interactions between these actors, produce an aggregate capable of achieving a desirable goal? What are the adaptive mechanisms that favour the emergence of desirable properties of the software-intensive acquisition?

Answering these questions is the essence of finding the success path for enterprises like software-intensive acquisitions and other similarly complex engineering projects.

The desirable emergent property is the system being able to achieve the intended goals, which would not be possible without the system, as for example when individuals work in isolation and without a common goal. The system may, and often does, develop undesirable emergent properties, as for example developing a product poor in quality, costing more than the allocated budget, taking more than the allocated time, and delivering a solution that does not satisfy the need. These undesirable properties are nevertheless emergent, as they would not exist without the system. At the end the system achieves its purpose, which may not be the intended purpose for the system.

Fitness Landscapes

The concept of ‘fitness landscapes’ has been applied to biological systems (Kauffman 1993, 1995), and to business organisations (Stacey 1999). The system’s fitness indicates how well the system is prepared to achieve its purpose. Biological systems that are able to adapt to changing environmental conditions are fit for survival. The same happens to business organisations. Companies that are fit would adapt to changes in the market and their competitors and find ways to continue in business. When the system changes its state and gets better prepared to achieve its purpose and survive changes in its environment, it moves to a higher level of fitness.

The fitness landscape represents all possible fitness states that the system can attain. Simple systems have their fitness dependent on a few and often independent variables. The fitness landscape of simple systems would be a smooth climbing terrain. The fitness of complex systems depends on many interrelated variables. Changing one of

these variables with the intention to increase fitness could cause an inverse, or sub-optimal, effect because that single change would affect other variables in the system. The fitness landscape of any system is determined by the strategies of the other system with which it interacts (Stacey 1999 p. 112). Stacey explains that the fitness landscape of a company in a competitive market is always changing in ways that is impossible to predict. A company introducing an innovative product in the market would trigger changes in the competition that could release a similar although better or cheaper product causing a reduction of fitness of the company that introduced the product in the first place.

Complex systems present a complex fitness landscape represented by a terrain filled with mountains, hills and valleys, referred as a ‘rugged fitness landscape’. Every state of the system corresponds to a specific degree of fitness corresponding to a position on the landscape. When the system is positioned in valleys or on low hills it is deemed to have a low fitness. Being positioned on higher mountains the system would reflect a higher degree of fitness. Due to the ruggedness of the terrain, and mountains being separated by valleys, to achieve improved fitness in the system would require a progression through states of lower fitness. Also, to climb to higher fitness will require some form of energy expenditure.

Companies in a commercial market quite often live in a rugged fitness landscape. Each peak may represent a local optimum but may not reflect the global optimum peak. The company spends resources to climb the terrain of the fitness landscape to achieve one of the local optimal places. Changes in the market change the landscape and companies will need to search for other local optimal positions. To do so the company will leave its current position and most likely will descend to a vale in order to climb another peak, and the journey will require spending resources.

Organisations that live in a rugged fitness landscape seek strategies to move to better fitness places as fast as possible and as cost-effectively as possible. Software-intensive acquisitions are no different. The acquisition is planned based on the information available at that time and after the proper assessment the acquisition is deemed to be fit to achieve its intended purpose. However, the system could be in a lower fitness position than initially thought because there may be many variables and possible situations that are unknown at the starting point.

As the acquisition progresses, more information is gathered and real situations are presented, revealing that the system is in fact positioned in a lower fitness place. A possible scenario is the realisation that the engineering resources lack the knowledge about the application and the experience to integrate a complex technical system. To get to a higher fitness position, engineers would have to spend time on training and learning and discussing the application with users and other experts from the customer's organisation. To achieve a higher mountain peak in the fitness landscape, time and financial resources need to be expended, and while engineers are preparing themselves for the task, their productivity for planned tasks will be reduced. These conditions leading to a higher fitness will also cause a temporary lower fitness, corresponding to the system going through a valley before reaching a higher mountain peak of fitness.

Edge of Chaos

Although the 'edge of chaos'¹⁶ can be described with mathematic rigour, it has become popular as a metaphor representing the boundary region of complex system states between order and disorder, stability and chaos. Systems that exist in a changing environment need to be continuously changing to be able to adapt to new conditions to be able to survive. Changing too much, the system may cross the boundary at the edge of chaos and collapse; changing too little the system may not be able to adapt to the new conditions that would secure its very existence.

The concept of 'edge of chaos' has been applied to biological systems (Kauffman 1993, 1995), and to business organisations (Stacey 1999). In both cases the conditions to evolve and survive lie at the edge of chaos, which is continuously changing.

The edge of chaos can be associated with the optimal state that contributes to the success of software-intensive acquisitions suggested by Flowers presented earlier in this chapter (see Section 2.1.1). To be successful the acquisition system has to be kept in a very specific state that would be able to execute what was planned, to deal with known risks and unknown situations, balance cost, as well as promote the conditions to develop and apply the variety required while maintaining cost and schedule. Complex software projects that rely too much on plans, schedules, processes and procedures become too rigid and are unlikely to be able to cope with what is unknown. DeMarco, once a strong

¹⁶ The term 'edge of chaos' was coined in 1990 by Christopher Langton (1990) while he was researching cellular automata computational techniques.

supporter of measuring and controlling the process of software development, acknowledges that not all in software development can be measured, much less controlled (DeMarco 2009). The Super Seasprite Helicopter programme was cancelled after 12 years of development and seven years of delays (DOD-AU 2008) because the programme under-estimated the complexity of the task and was unable to cope with changes in safety and airworthiness requirements (Ferguson & Blekin 2008). These projects need some flexibility to allow information to be gathered and shared, specifications to change, and plans to be modified so that the project can achieve what is needed. However, if the project becomes too flexible and loses sight of its goals and objectives it is likely to end up in failure. The optimum state referred as the ‘edge of chaos’ should be reached and maintained by experienced managers, leaders and workers.

2.3.4. Structure Influences Behaviour

Senge discusses the implication of systemic structure and asserts: ‘when placed in the same system, people, however different, tend to produce similar results’ (Senge 2006 p. 42). Systemic structure is not published in organisational charts and is created by conditions that drive beliefs and motivations that make the people in the system behave in certain ways. According to Senge (2006 p. 44), the interrelationship of key variables in the system creates a systemic structure that determines the behaviour of the system over time. The behaviour of social systems is determined by how people behave; therefore people are part of the systemic structure. What people in the system often don’t realise is that they can influence and change the system structure by changing their own behaviour and as a consequence the behaviour of the system. In fact, structure influences behaviour which in turn influences structure that again influences behaviour ‘ad infinitum’. Therefore, the system could be in a permanent state of change searching for higher peaks in its fitness landscape, as discussed in Section 2.3.3.

This thinking has profound implications in managing the social system of software-intensive acquisitions. By understanding the systemic structure managers and leaders could change their own behaviour and make decisions that would change the behaviour of others in the system and of the system itself. The systemic structure of software-intensive acquisitions will be further discussed in Chapter 3.

2.3.5. Complex Adaptive Systems

The study of complex adaptive systems (CAS) originated also at the Santa Fe Institute led by scientists such as Murray Gell-Mann and John Holland. CAS comprised interconnected components, often called agents, which have the capacity to change behaviour due to changes in the environment and the interaction with other agents.

Intelligent or non-intelligent agents can form CAS. Biological systems formed by cells and non-intelligent organisms are capable of adapting through genetic selection. Agents that are better fit to survive in the environment have better chances to procreate and their offspring will carry the genetic characteristics of the parents. The population of agents changes with time in such a way that future generations will have been genetically adapted to the conditions presented by the environment. Intelligent agents that change behaviour and their actions through learning from past experiences form social CAS. Adaptation occurs by learning what works and what does not, and then selecting behaviours that lead to better results, often a process of trial and error. Social systems exist in the animal world with a vast range of agent intelligence: from insects, birds and mammals to human beings.

The definition of CAS varies from one scientist to another. While Holland (1995) calls a system a CAS only when it is a collection of interacting ‘adaptive agents’, Gell-Mann considers ‘adaptive agents’, also complex adaptive systems, a system of systems, or a CAS of CAS (Gell-Mann 1995).

Seven Basic Characteristics of CAS

Holland (1995) proposes seven basic characteristics that are common to all CAS regardless of the nature of the system, comprising four properties and three mechanisms, as shown in Table 3.

Table 3 – Seven Basics Characteristics of CAS

Properties	Mechanisms
Aggregation	Tagging
Non-Linearity	Internal Models
Diversity	Building Blocks
Flows	

The seven basic characteristics of CAS will be presented and illustrated with examples of social systems.

Aggregation

Aggregation is a central property of CAS and enables interaction between the agents. Without aggregation there is no interaction and agents in isolation do not constitute a system. Software-intensive acquisitions are aggregations of organisations that are connected by contracts. Organisations aggregate people into departments, groups and teams forming purposeful task forces to fulfil the contracts.

Aggregation of simpler agents can create more complex structures that could be seen as meta-agents. Social systems contain individuals aggregated in groups and teams, and groups and teams aggregate into more complex structures as in organisations. Teams, groups and organisations are meta-agents of more complex systems, like business enterprises, software-intensive acquisition and the entire market place. People, although very complex agents, are simpler agents than meta-agents in social systems. There are no fundamental differences between agents and meta-agents; it is just a matter of chosen level of granularity.

Tagging

Tagging is the mechanism that allows agents to identify other agents of certain types that are then selected to aggregate and interact. When an agent is assigned to a team it is tagged as belonging to the team and becomes a candidate to interact with members of that team and other agents in the system that have an interest in that type of tag. Often an agent is tagged as part of several meta-agents: the team, the group, the organisation, and the software-intensive acquisition program. Agents belonging to the engineering team, and as such carrying that tag, are candidates to interact with agents that have that same type tag in other groups and organisations. Role, responsibility and speciality are formal tags in engineering projects.

Tags are also created informally in social systems through relationships and acquaintances. Agents are more likely to interact with agents they know because they have positive expectations from the interaction. Agents may avoid interacting with agents they don't know or that have a reputation, also a tag, of being hostile. Informal tags create networks of informal communication that are always present in social systems and engineering projects.

Internal Models

‘Internal models’ is the term adopted by Holland to describe the mechanism that makes agents do what they do as individuals in the system. Internal models drive the agents to observe and assess the environment, communicate with other agents and make the decisions that determine their actions. Non-adaptive agents have fixed internal models, while adaptive agents change these models as they interact with a changing environment, interact with other agents, accumulate experience and learn. People are the adaptive agents in software-intensive acquisitions and their internal models are constructed based on their professional training, experiences and beliefs as well as the processes and procedures adopted in the system. Adaptive agents will be discussed further in this section (see Section 3.5.1).

Building Blocks

‘Building blocks’ are part of a mechanism in CAS that enables reuse. Building blocks are pervasive in the way agents adapt and develop internal models. Complex systems often present situations never experienced before. Intelligent agents are likely to use and combine rules that were used successfully in the past in similar situations. The use of building blocks is discussed further in this section together with adaptive agents (see Section 3.5.1).

Building blocks are also a concept used to design systems and CAS. Man-made social and technical systems often contain proven building blocks combined and even modified to attend different needs. People’s professional training and experience provide the building blocks to construct new artefacts and deal with new situations in software-intensive acquisitions.

Non-linearity

Non-linearity is a property of CAS seen as the collective response of the system, as a whole is not the same as the response of the sum of its components. Although this is a common characteristic of complex systems, CAS make non-linearity even more present as agents change and adapt to changes in the environment and the interaction with other agents. Holland attributes non-linearity in CAS to the agent’s internal models, which change as the agents adapt.

Non-linearity in software development projects was reported more than three decades ago by Frederick Brooks in his classic ‘The Mythical Man-Month’ (1995), suggesting that productivity and consequently the duration of the project is not proportional to the number of people allocated to the project and that assigning more people to an already late project causes even greater delays. Non-linearity in this case, as explained by Brooks, is caused by the time spent to learn the job and the communication between software developers and management.

The consequences of the ‘mythical man-month’ can be correlated with, and better explained by, the characteristics of CAS. Changes in the environment of already late projects, caused by the addition of new agents in the system, and likely increase in management pressure, causes the agents, i.e. software developers and management, to adapt to the new conditions. Adaptation will change their internal models to cope with additional communication, learning and teaching newcomers and working under additional pressure. The change in internal models causes an opposite effect to what is desirable and was expected. As the result the system develops new and undesirable emergent properties reflected in lower quality of the software product, lower productivity and consequent further delays.

Diversity

Diversity is a property of CAS and the agents in the system are the source of diversity. The diversity of agents is the source of variety to compensate the variety originated in the environment or within the system itself in the form of the multitude of unforeseen situations that can eventuate in complex systems.

Systems that are capable of adapting to new conditions, and maintaining their very existence, need diversity to compensate for unforeseen or unplanned changes. The rainforest is an example of an extremely robust natural system. The diversity of animals and plants provide the variety that makes the system so robust to changes in the environment and natural resources. Diversity and the resultant robustness come with a price, that of lowering the efficiency of the system, as suggested by Hitchins (1992).

Social systems are not different from other systems as far as diversity is concerned and people, as the agents in the system, are the source of diversity. As the complexity of business enterprises, engineering projects and software-intensive acquisitions increase,

the diversity of people in the system has also to increase to compensate for unforeseen and unplanned events. People diversity should provide ‘requisite variety’ (see Section 2.3.1) in the form of useful knowledge and experience and teamwork behaviour. A software-intensive acquisition as in other sorts of engineering projects, requires a multitude of professionals with a wide range of specialties and experience. The success of software-intensive acquisitions does not depend only on software engineers. Software engineers are necessary to solve problems and perform tasks within the software domain. Other kinds of agents are also needed to perform other tasks in the software-intensive acquisition system: users, engineering experts, managers, technicians, secretaries and other support agents. Quite often the system will need agents with specialities and experience in areas that have not been accounted for. This apparent excess of diversity may be the difference between success and failure in achieving the intended goals.

In the same way as for the rainforest, when the diversity of business enterprises increases, the system becomes more robust and the chances of success are improved. However, efficiency decreases, potentially causing schedule delays and the reduction of profit margins. The trade-off between higher diversity and robustness lowers efficiency and is a contradiction that creates a management dilemma. As suggested by Stafford Beer (see Section 2.3.1), managers often choose to disobey the Law of Requisite Variety, and naively choose to attempt systems that should be more efficient and profitable. At the end, the system does what it does best, and if the system is not structured to do what it is intended to do, failure is inevitable.

Flows

‘Flows’ is a property of CAS that reflects the movement of what agents exchange when they interact or interact with the environment. Depending on the nature of the CAS what flows varies greatly: the flow of goods and money in the market place; the flow of ideas in a social system; and the flow of food in an ecosystem. Organisational communication carries the flow of expressed needs, knowledge, problems and solutions in engineering projects. Organisational communication impacts actions and relationships and is influenced not only by the communication process but also by cognitive-affective factors (Te’eni 2001). Where there is interaction there is a flow of some sort, whether explicit or implicit. The more intelligent the agents, the more complex are the

interactions and flows. Identifying flows is an important aspect of understanding CAS and in particular social CAS where there may be different flows.

2.3.6. Framework for Harnessing Complexity

The ‘Framework for Harnessing Complexity’ (Axelrod & Cohen 2000) is not intended to control complex systems. Instead, the framework provides guidance to steer the system similarly to the way a harness steers a horse without necessarily controlling it. The guidance comes in the form of an understanding of the underlying processes that drive the system. The framework comprises three key interlocked processes: *variation*, *interaction* and *selection*.

The framework applies to CAS where agents interact with other agents, with artefacts and with the environment. Agents have cognition while artefacts have not. This research will use the term ‘actor’ for cognitive agents and ‘artefact’ for non-cognitive agents. Actors and artefacts are agents in the system. People are actors in a social system. Artefacts are objects that actors use, make or transform. Actors are the adaptive agents in the system.

Population represents a set of actors, artefacts or strategies that share some common characteristics. Actors in a population do not necessarily have to interact, as expected in aggregations. Population can be used to specify the set, or population, of engineer actors in the system, or the population of strategies that foster learning. Actors in the same population and considered to be of the same type have the same or similar strategies. In the case of actors that represent people, variations on similar strategies can be associated with attributes of the actor such as personality and behaviour patterns.

Variation in the context of the framework is equivalent to the combination of the concept of variety in Cybernetics and variation in the System of Profound Knowledge and represents the diversity in the population of actors, artefacts and strategies in the system.

Interaction is what happens when agents (actors and artefacts) in the system relate or communicate to each other. Actors interact with other actors and with artefacts. Interaction between actors can serve to assign tasks and reports progress and may cause changes in both actors in many different ways: increase knowledge and experience; increase or decrease motivation; cause the actors (or one of them) to be happy or upset;

or induce several other changes of behaviour. Actors interact with artefacts as the end goal of tasks assigned to the actor to create or modify the artefact; or perhaps to use the artefact as a tool or source of information. An artefact can also interact with other artefacts in a form of passive interaction. Artefacts may have a functional association or a dependency (e.g. software components may work as part of a larger system or may have a dependency on specification documents).

Strategy defines the way that an actor responds to its surroundings and pursues its goals. The concept of strategy is similar to what Holland calls internal models. Actors apply some ‘measure of success’ to assess their strategies and consequent performance. Measures of success are used by the actors to execute the ‘credit assignment’ process and assign credit to their rules and strategies to ensure they perform in accordance with the measures of success. Actors often have explicit and implicit measures of success. Explicit measures of success are associated with what is visible: compensation, recognition, meeting deadlines, quality of the work, etc. Implicit measures of success are not always visible and are associated with the agent’s feelings and personal goals: enjoying the work, increasing knowledge and experience, contributing to the project with one’s own ideas, being part of a constructive and successful team. Both kinds of measures of success contribute to the way actors select strategies and adapt.

Selection is the process that causes good strategies to be applied more often while bad strategies will be tested and discarded. Actors choose strategies that have a better chance to achieve their goals. Strategies that produce good results will be applied again and may be even improved. Successful strategies are copied by other actors and become popular. Unsuccessful strategies are discarded and disappear. An actor may ask help from another actor. If the response is satisfactory the actor identifies that asking for help of that particular actor is a good strategy, which may be communicated to other actors who will likely adopt it. On the other hand, if the request for help is refused the strategy is discarded and dies out.

By understanding the processes of variation, interaction and selection, systems deemed to be complex and adaptive could be better understood. The Framework for Harnessing Complexity will be applied to develop a model for software-intensive acquisition later in Chapter 5.

2.4. Psychology and Social Behaviour

Psychology, in the context of this research, is what scientists and philosophers of various persuasions have created to try to fulfil the need to understand the minds and behaviours of human beings¹⁷. This research is interested in the human behaviour that contributes to success or failure of software-intensive acquisitions.

Software-intensive acquisition is a problem-solving group task, performed by people associated in teams and organisations working together to achieve a common goal. The tasks often are not completely defined, are intellectual, labour intensive and require creativity to be accomplished. Therefore, knowledge of psychology is intended to understand the behaviour of individuals working in social environments like a group, team and organisation. Within this context, managers need knowledge of how people are motivated to execute their tasks; how people learn and teach, cooperate and compete, lead and follow; how ideas are shared and mind concepts are formed; and how collective knowledge and intelligence are created. Management is about managing people and for effective management, knowledge of psychology is essential.

There are a vast number of fields of interest for this research within the context of psychology and sociology, many of them called sciences: sociology science, behavioural science and organisational science. It is important to note that science implies a causal explanation so that experiments can be repeated under the same conditions. For this research, and without wanting to create a polemical discussion, the field of so-called social sciences will be treated as social theories in the way that these theories can provide effective prediction without the certainty that sciences provide. Knowledge of psychology and other social theories helps to understand the most important component of the software-intensive acquisition system: people. Social theories presented here will be used to develop a model of software-intensive acquisitions.

The acquisition and development of novel and complex systems often starts with vague descriptions and incomplete specifications. Ideas are shared, complemented and developed into specifications and designs to become reality. Even detailed documents are not always complete and often carry defects that have to be revealed and corrected.

¹⁷ Definition adapted from the ‘Dictionary of Psychology’ (Reber & Reber 2001).

When the system depends on software components, the process of transferring and developing mental pictures continues even longer. All these activities are carried out by people and depend on how people are motivated to learn and cooperate to do their jobs.

In addition to the technical tasks described in the previous paragraph there are many support tasks that influence the development of the system. The acquisition of novel and complex software-intensive systems most certainly involves contracts between customer and suppliers; the prime supplier often purchases equipment and assigns part of the work to sub-contractors. All these relationships are commercial in nature, defined and controlled by contracts and schedules. As with the definition and specification of the system, contracts are not always perfect and lead to issues to be addressed and resolved during the course of the contract. These issues are often caused by gaps in the technical specification, not yet resolved or understood, and may lead to conflicts of interest. In the centre of all this is the figure of the manager, who is responsible for negotiating the best deal for the customer or supplier organisation he or she represents. Knowledge of psychology and social theories can be of use during contract negotiations and for stakeholder management.

Management occurs at many levels in the organisation. From the CEO to the project manager and the team leader, all are managing people to do their jobs even when the role or job is not completely defined. The task of the manager is to motivate people to do what they are required to do to achieve the best result for the individual, the organisation and the acquisition as a whole. For that to occur, knowledge of psychology and social theories is essential.

This section introduces concepts, definitions and theories that are used throughout this thesis. Some of the definitions are extended to ‘virtual agents’ and are applicable to modelling and computer simulation developed by this research. The discussions will refer to managers and developers. Managers and developers are commonly referred to as professionals. Managers are dedicated to manage the process and are not directly involved in developing the product. Developers are engineers and technicians involved in technical tasks to develop the product.

2.4.1. Cognition, Emotion Perception and Motivation

The definitions that follow, also included in the Glossary of this thesis, were adapted from the Dictionary of Psychology (Reber & Reber 2001) to fit into the context of this research.

Cognition refers to activities as thinking and reasoning and is associated with any class of mental ‘behaviours’ leading to the mental picture of ideas, pieces of knowledge, complex concepts and problem solving. Cognitive process leads to rational thinking.

Emotion is an umbrella term for the state created by emotions such as love, hate, fear, terror, good and bad feelings, etc. Emotional state is not the result of rational thinking. Feelings, as one of the dimensions of emotion, can lead to hypothesis developing beliefs that are not supported by real evidence.

Perception comprises those processes that give coherence and unity to sensory input, being physical and emotional. Perception is the person’s interpretation of a situation, being an awareness of what the person believes is the truth. Perception can be the result of a rational and emotional process.

Motivation is the intervening process or internal state of a person or a virtual agent that drives their behaviour or impels them into action.

2.4.2. Theory of Motivation

Understanding what motivates people helps to create the conditions to motivate them to perform well in doing what they are supposed to do. A theory of motivation can be used to model human behaviour. This research aims to apply a theory of motivation to construct a model of software-intensive acquisitions.

Maslow’s ‘Hierarchy of Needs’ (Maslow 1943) offers a theory of human motivation driven by five levels of needs: ‘physiology’ is the most basic of human needs, followed by ‘safety’, ‘love and belonging’, ‘esteem’ and ‘self actualisation’. The hierarchy of needs implies that a person seeks first to obtain the lower level needs and when these are satisfied, higher needs become important. Physiology includes breathing, food, water, sleep, shelter and other physiological needs. Safety is about security of the body, health, employment, family, property, and the like. After physiological and safety needs are satisfied, a person seeks friendship and family ties to fulfil love and belonging

needs. The top two needs are about self-achievement. Self-esteem, self-confidence, personal achievements, self-respect and being accepted by others are part of the esteem need. Self-actualisation is the need to achieve everything else one is capable of achieving: creativity, problem solving, acceptance of facts, morality, and others. Seeking to obtain a higher education degree is a typical case of satisfying the two top levels of the hierarchy of needs.

The Self-Determination Theory (SDT) (Ryan & Deci 2000) suggests that there are different types of motivation based on different reasons that drive a particular action. Ryan and Deci indicate that the most distinctive types of motivation are *intrinsic motivation*, which is associated with performing an activity because it brings satisfaction rather than a reward or consequence, and *extrinsic motivation*, which is the driver for performing an activity in order to attain a separate outcome (Ryan & Deci 2000 pp. 55-56 & p. 60).

What motivates managers, engineers and software developers? The hierarchy of needs suggests that lower level needs should manifest earlier and when these are satisfied the person will seek higher level needs. One could expect that senior professionals would be motivated by the needs of esteem and self-actualisation, while junior professionals would concentrate on safety and belonging. This assumption is not always the case and both junior and senior professionals can experience a combination of lower and higher level needs. The need for safety and employment security, for instance, is identified as a lower level need and it is often a motivation driver for senior professionals who seek stability at later stages in life and may not be as important for junior professionals. It is understood that the need for job security may change considerably during times when the employment market is low.

It is reasonable to assume that senior professionals will be more motivated by the need for esteem, self-respect and being respected by others, than do junior professionals, who will seek to be part of a team and the organisation, motivated by the need to belong. The need for self-actualisation is present in a broader spectrum of professional experience. Challenging tasks that require creativity and problem solving skills often motivate senior and junior professionals. Junior professionals, however, have a stronger need to do new and challenging things, perhaps because they have not experienced a range of challenges before. Senior professionals most likely have experienced many challenges,

and give more emphasis to other aspects of self-actualisation such as the acceptance of facts and morality.

Managers and developers have different, and in some cases, conflicting motivations (McConnell 1996). Managers are motivated by responsibility for meeting cost, schedule and contractual obligations; developers are motivated by the challenge to develop high quality products that require learning, new technologies and creativity (Linberg 1999; McConnell 1996; Procaccino, Verner & Lorenzet 2006). If cost, schedule and contractual specifications are consistent with what the developers' judge needs to be built, there is no conflict and the project will succeed. In reality this is quite often not the case, and creates tensions between managers and developers. The source of such tensions can be explained by the conflict of motivations.

Aslaksen explains that project management and engineering management operate in different domains, physical and functional, respectively (Aslaksen 1996 p. 41). While project management deals with physical entities like tasks, costs and schedules, engineering management deals with functional entities, such as requirements and design that are strongly supported by abstract concepts. Aslaksen indicates that based on his experience, managers and engineers fail to grasp the distinction between project management and engineering management. The Aslaksen contribution supports the conflict of motivations between managers and developers and the origin of tensions that may arise between them.

Software developers can make or break a software development project and knowing how to motivate them is the key for success (McConnell 1996). Developers are often motivated by the task itself, as a form of intrinsic motivation, although they expect some form of recognition when they perform well and beyond their duties. Rewards, as a form of extrinsic motivation, can be used as incentive and recognition. Rewards, when used to increase the motivation of managers and developers, often seem to motivate the attainment of the reward and not necessarily to do what is truly needed to achieve the intended goal (McConnell 1996).

2.4.3. Theory of Behaviour

Behaviour is any noticeable change, response or action of a person, a virtual agent or system. Behaviour is driven by motivation resulting in action intended to fulfil a need.

The theory of behaviour aims to provide ways to predict the likely behaviour of a person in accordance with a classification of personality.

Theory of Behaviour in Context

This research is interested in the kinds of behaviours that contribute to the success of the acquisition and those that move the acquisition from its objectives. The acquisition system will behave as the result of the collective behaviour of individuals. The behaviour of the system is not expected to respond linearly to individual behaviour, as one individual may influence the behaviour of others, and can even feedback and change the behaviour of the individual that started the process of change. A theory of behaviour is of fundamental importance to create models of people as individuals as well as associated in groups, teams and organisations. This research aims to apply a theory of behaviour to construct a model of software-intensive acquisitions.

Managers and developers, driven by their own motivations, may have different perceptions of what kind of behaviour will be best for achieving success. Managers believe that developers should behave to perform the tasks assigned to them, meeting the contract in accordance with the established plan. Managers want the developers to behave efficiently to reduce costs and increase profit margins, and avoid behaviours that result in unplanned tasks. Developers often believe that their behaviour should lead to developing a product that is technically sound and will meet the need of the customer, even resulting in additional cost and schedule overrun. Developers also often believe that managers should support what is needed to support their behaviour.

A detailed plan can be a realistic approach for simple projects, but often is a naïve approach for complex tasks. It is a fact that cost, schedule and contractual specifications are important, yet these may be unrealistic. Both managers and developers can be right or wrong depending on the situation. When specifications are incorrect or incomplete, or if costs and the schedule are incompatible with the task and resources, or the available expertise is not adequate to develop the system, behaviour that brings about creativity, learning and cooperation would be more appropriate. Nevertheless, large differences between the plan and reality would impact costs and the schedule. The right kind of behaviour can reduce these impacts.

Behavioural science and its theories is a vast field. It is not the object of this research to perform a deep investigation of the subject, but to find theories that can be applied to

model people, teams and organisations within the context of software-intensive acquisitions as discussed above.

2.4.4. Personality Types

Several theories exist to classify people in accordance with personality types. None of these have been scientifically proven and remain as theories of practical use. Personality types help to describe personal characteristics and predict how a person may behave and perform under certain circumstances. Therefore, the theory of personality types can be useful for modelling human behaviour. This research aims to apply a theory of personality types to construct a model of software-intensive acquisitions.

Three theories have been considered: Myers-Briggs Type Indicator (MBTI), Keirsey Temperament Scale (KTS) and the Life Styles Inventory (LSI).

MTBI and KTS

The Myers-Briggs Type Indicator (MBTI) (Briggs & Myers 1980) is an expansion of Carl Jung's Psychological Types (Jung 1971). MTBI assessment uses a psychometric questionnaire that evaluates people's psychological preferences for each attribute. The assessment places the person on one of the sixteen MBTI types and gives a quantitative score to each attribute to indicate how well the person fits the type and how strong is the presence of each attribute. MBTI is used in team building, group dynamics, organisational culture, personal counselling, coaching and development.

The Keirsey Temperament Scale (KTS) (Keirsey & Bates 1984) is a self-assessment that generates a personality classification based on four temperaments as defined by Plato: Artisans, Guardians, Idealists, and Rationalists. Each temperament can be further classified into four possible roles, and each role has two variants. The combination of temperament, role and variant produces sixteen possible types. There is a correlation between KTS and MBTI types (Keirsey & Bates 1998).

Linberg (1999) applied the four KTS temperaments to classify software developers to investigate how software developers perceive success and failure of software projects. According to Linberg, software developers classified as Artisans thrive on chaos and ambiguity because they like challenges; Idealists tolerate ambiguity and don't like chaos; Guardians dislike the ambiguity of the early phases of projects and wish that

everything was well defined; Rational developers prefer innovative projects and like exploring solutions, and get bored after the problem is solved.

Life Styles Inventory

The Life Styles Inventory (LSI)¹⁸ (Lafferty¹⁹ in Cooke & Rousseau 1983) is a personal psychological assessment of thinking and behavioural styles. The assessment is in two parts: the first, LSI Level I, is a self-assessment; the second, LSI Level II, is an assessment performed by others, producing a classification as the person is perceived by peers, subordinates and superiors. LSI focuses on four areas of concern from the point of view of the person being assessed: task/satisfaction; people/satisfaction; task/security; and people/security. The LSI four-factor model combines Maslow's theory of needs with person-centred versus task-centred models of leadership (Cooke & Rousseau 1983). LSI assesses twelve life styles attributes that form a continuum correlated with the four areas of concern and three characteristics of behaviour, shown in Table 4.

Table 4 – LSI Attributes

Area of Concern	Life Styles		Interaction Style Classification
People and Satisfaction	1	Humanistic-Helpful	Constructive
People and Satisfaction	2	Affiliative	Constructive
People and Satisfaction/ People and Security	3	Approval	Passive
People and Security	4	Conventional	Passive
People and Security	5	Dependent	Passive
People and Security/ Task and Security	6	Avoidance	Passive
Task and Security	7	Oppositional	Aggressive
Task and Security	8	Power	Aggressive
Task and Security/ Task and Security	9	Competitive	Aggressive
Task and Security	10	Competence (Perfectionist)	Aggressive
Task and Security	11	Achievement	Constructive
Task and Satisfaction / People and Satisfaction	12	Self Actualisation	Constructive

¹⁸ Life Styles Inventory (LSI) is a Trade Mark and copyrighted by Human Synergistics.

¹⁹ Lafferty, J. C., 1973, Human Synergistics evaluation system Level I: Life Styles, Plymouth, MI: Human Synergistics, 1973.

The information in Table 4 is usually represented graphically as a circle, the Circumplex²⁰, divided into twelve sectors corresponding to the twelve life styles, so that life style 1 (Humanistic-Helpful) is side by side with life style 12 (Self Actualisations). Life styles that are close to each other on the Circumplex are more correlated than life styles that are far apart. Self Actualisation and Humanistic styles, for example, are associated with the need for self-actualisation that is the top level of Maslow's 'hierarchy of needs' and are thus correlated.

The life styles are also classified in accordance with their interaction style as Aggressive, Passive or Constructive, represented respectively by the colours red, green and blue.

The theory of interaction styles applied to the dynamics of group interaction, discussed further in this section, can be useful in modelling human behaviour in teams and organisations, and will be applied to the model of software-intensive acquisitions presented in Chapter 5.

2.4.5. Groups, Teams and Organisations

The term 'group' is used in the context of 'social group', where the members of the group are persons classified together on the basis of some social factor and there is a degree of interdependency between the members of the group²¹. Team is a group where the members of the group perform a common task and the contribution of each individual is important to complete the task. This research will use both terms 'group' and 'team' interchangeably to mean 'team'.

Organisations are social systems formed to create products and/or services of value to others (Rechtin 2000). Organisations may contain several groups. Companies are organisations that have as an objective, some form of business.

Organisations are CAS

Organisations are complex adaptive systems and creative organisations operate at the edge of chaos (Stacey 1999). The concept of edge of chaos, introduced in Section 2.3.3, is defined by Stacey as: 'a form of bounded instability found in the phase transition

²⁰ The Circumplex, as the LSI, is proprietary of Human Synergistics and has been omitted for copyright reasons. The information here included is supported by the reported references. The Circumplex can be found at <http://www.human-synergistics.com.au/content/products/circumplex/Default.asp>.

between the order and disorder zones of operation for a complex adaptive system'. Contradictory forces (e.g. cooperation and competition; known and unknown; safe and risky; save and invest; etc) that are continuously being rearranged, drive this state of operation and prevent the system from being trapped in a local optimum of the fitness landscape (Stacey 1999 p. 102). The state of permanent change that keeps the organisation at the edge of chaos is a realisation that structure influences behaviour and behaviour influences structure, as discussed in Section 2.3.4.

Operating at the edge of chaos relies on emergent behaviours that cannot be predicted; intentional and rational processes are ineffective for planning action and long-term outcomes; and it does not guarantee success, as it all depends not only on what the organisation can do, but also on what other systems that interact with the organisation, are doing. There is no easy solution. When operating in a complex fitness landscape, the organisation may opt for safety and stability or take chances with more bold strategies that would place the organisation at the edge of chaos. The first will keep the organisation still and most likely in a position of self-destruction; the second will give the organisation a chance of survival. The choice may not be obvious without seeing the system from the outside.

The fitness landscape of innovative software-intensive acquisitions is also complex, and where multiple organisations operate having common and conflicting interests. The customer organisation aims to lower the cost-benefit of the solution, i.e. get more for less, while the objective of the provider organisation is to maximise profitability albeit achieving customer satisfaction. When the acquisition is well defined, estimated and planned, and matches the capabilities of the provider organisation, the complexity of the landscape is reduced. This scenario is rather the exception than the norm.

Complex software-intensive acquisitions present the characteristics of complex adaptive systems operating in complex fitness landscapes. It is therefore expected that the complex acquisitions will have to be managed at the edge of chaos to create emergent behaviours that would steer the enterprise to successful or at least acceptable outcomes. In Chapter 3, the software-intensive acquisition system will be mapped to the seven basic characteristics of CAS (as outlined in Section 2.3).

Desirable Emergent Properties of Organisations

Organisations depend on emergent properties to be able to achieve their objectives. Rechtin suggests three emergent properties that add value to organisations: quality, knowledge and core competency (Rechtin 2000). Quality is the result of collective effort and emerges when the organisation is effective in what it does. Knowledge emerges as the collective knowledge of the organisation. Collective knowledge comprises explicit and implicit knowledge and is greater than the sum of the knowledge that existed before the organisation was formed. Core competency is what makes the organisation more than buildings and processes, and embodies the culture and the capacity of the organisation to achieve its objectives. Core competency resides in each individual member of the organisation. These emergent properties will be discussed within the context of software-intensive acquisition in Chapter 3: Section 3.6.

Stafford Beer's insight about the purpose of the system helps to understand the implications of emergent properties of the system: 'the purpose of a system is what it does' (see Section 2.3.1), and what the system does is the result of its emergent properties.

Emergent properties are difficult to predict and therefore also difficult to include as an attribute of the organisational design. Emergent properties come into being only after the organisation is functioning, and they change as time progresses and the system adapts to new conditions.

The aim of designing systems whether simple or complex is to find architectures that will enable the system to achieve the intended purpose. The design should enable the system to develop the emergent properties that would make the system do what it is intended to do. The use of the term 'enable' means that there is no guarantee that the desired properties will emerge. The best that can be done is to attempt to steer the system in the desired direction, which requires a good understanding of the system. This approach is the essence of the Framework for Harnessing Complexity presented earlier in Section 2.3.6.

The goal of designing organisations is therefore to create structures that will enable the organisation to achieve its intended purpose. The design of software-intensive acquisitions is thus to create structures that allow the organisation to identify and

communicate the need; define the problem; solve the problem and implement the solution that satisfies the need within the constraints of time, cost and quality.

Although simple to enunciate, designing organisations for an intended purpose is a challenge without easy solution. It is reasonable to say that desired emergent properties should add value to the organisation in achieving the intended goal.

2.4.6. Social Behaviour in Groups, Teams and Organisations

Social behaviour is any behaviour of an individual that has a social component; it is the behaviour that influences others and is influenced by the presence, attitude and actions of others²¹. People in teams and organisations are the subjects of social behaviour.

Impact of Group Interaction Styles on Problem-Solving

Problem-solving groups work as a unit to solve a common problem, as in software-intensive acquisitions and engineering projects. There are two principal objectives of problem-solving groups: to achieve high quality results; and to promote acceptance of the solution (Hoffman 1979). To achieve the desired quality requires the maximum utilisation of resources brought by each individual member of the group. To obtain acceptance of the solution implies a high level of motivation for carrying out the group's decision. These objectives create pressure on each member of the group to decide between bringing their unique, and possibly controversial, contribution to the problem-solving task, or to conform to a group's decisions.

The effectiveness of problem-solving groups depends on the group interaction style (Cooke & Szumal 1994), which defines the style of the group as a whole unit, as a system, resulting from the dynamic interaction of its members in accordance with their roles and individual personality styles. The group interaction style reflects the atmosphere of the group, the group's own personality that will promote cooperation or persuasion in finding the solution for the problem. Group members with more power, e.g. in management or leadership roles, or with stronger personality styles are likely to influence other members in the group (Cooke & Szumal 1994). Across problem-solving groups, solution quality increases when the group shows a constructive interactive style and decreases with a passive interaction style; and the acceptance of the solution increases with a constructive interaction style and decreases with both passive and

²¹ Definition adapted from the 'Dictionary of Psychology' (Reber & Reber 2001).

aggressive interaction styles. Aggressive interaction was shown to be unrelated to the quality of the solution (Cooke & Szumal 1994).

Group Styles Inventory

The Group Styles Inventory (GSI)²² is an instrument in the form of questionnaires to assess group interaction style, that is, ‘the ways in which members interact with one another and approach the task to be accomplished’ (Cooke & Lafferty²³ in Cooke & Szumal 1994). The GSI classifies group interaction styles as Aggressive, Passive and Constructive.

Groups with a constructive style are more likely to reach a balance between individual and shared interests and achieving both individual and group objectives. Constructive groups tend to maximise the contribution of each group member by fostering cooperation between the group members, and reaching decisions through cooperation rather than by persuasion.

Groups with a predominant passive style reflect the preference of their members for people-oriented behaviours to fulfil their needs of security and acceptance. The members of the team are concerned with people well-being and adopt a position of conformance and conflict avoidance, even in constructive discussions. Information sharing is limited and resistance to delivering bad news is a natural choice.

In the aggressive group style, members are task-oriented and place the task objectives ahead of people well-being, behaving in ways to promote their status and position to fulfil their own needs. The members of the group have a tendency to demonstrate their competence by doing things perfectly, without seeking help or helping others and often losing sight of the group’s goals.

Dynamics of Group Interaction

Cooke and Szumal (1994) correlated group interaction style with the dynamics of group interaction and established that in problem-solving groups, solution quality increases when the group shows a constructive interactive style and decreases with a passive interaction style, while the acceptance of the solution increases with a constructive interaction style and decreases with both passive and aggressive interaction styles.

²² Group Styles Inventory (GSI) is a Trade Mark and copyrighted by Human Synergistics.

²³ Cooke, R. A., Lafferty, J. C., 1988, Group Styles Inventory, Plymouth, MI: Human Synergistics.

Aggressive interaction has been shown to be unrelated, or marginally positive, with respect to the quality of the solution.

The dynamics of group interaction styles tells us that constructive behaviour promotes constructive behaviour, and aggressive behaviour suppresses constructive and promotes passive behaviours; that is aggressive behaviour promotes passive behaviour altogether (Cooke & Szumal 1994).

The impact and dynamics of group interaction styles to problem-solving groups is important for the success of software-intensive acquisitions and will be discussed in Chapter 3.

Leadership Styles

Bass (in Turner & Müller 2005) identified two types of leadership styles – transactional and transformational. Transactional leadership style is task-oriented, rewards followers for meeting performance targets and manages by exception when problems occur. Transformational leadership style is people-oriented, challenging followers with new ideas and motivating them by developing a vision and appropriate behaviour. The objective of transformational leadership is therefore to transform the individual and the organisation to achieve higher effectiveness.

Berson and Linton (2003) performed an empirical study of 511 engineers in the telecommunications field and found that both transactional and transformational leadership are related to quality in the research and development (R&D) environment. However, transformational leadership was found to support the quality of the R&D environment and employee satisfaction more than the transactional leadership style.

Thite (1999) expanded the leadership model proposed by Bass by defining the technical leadership style, as one that relate the role of a leader as a catalyst. The study carried out by Thite indicated that the combination of transformational, technical and transactional produces better results in IT project management. According to Thite the practice of transactional leadership alone would lead to negative consequences. However, transformational leadership, as the main leadership style, can achieve better results when combined with a component of transactional leadership. Thite's findings suggest that constructive-learning-collaborative environments are effective as long as the task to achieve the target objectives is kept under control.

A project manager's leadership style can be linked with project schedule and cost performance (Yang & Chen 2010). Yang and Chen reported that higher levels of transactional and transformational leadership are associated with higher levels of project schedule and cost performance and that transformational leadership shows higher association with project performance and success over transactional leadership, although the authors do not specify the complexity and type of the project. Yang and Chen also reported that team performance, encompassing team communication, cohesiveness and collaboration, could mediate the influence of leadership style over schedule and cost performance.

Johnson and Klee (2007) researched the influence of leadership styles on passive-aggressive behaviours. In addition to transactional and transformational leadership styles, the authors included the autocratic style representing the leader that demands the obedience of followers to conform to directives of the leader, using coercion to achieve a common goal. Johnson and Klee define the passive-aggressive as a cluster of behaviours that convey aggressive feelings through passive means including: action avoidance, over-conforming, resisting change, indirect and physical passive behaviours, among others. The research results indicate that an autocratic leadership style would encourage passive-aggressive behaviour more than the transactional and transformational styles.

The results reported by Yang and Chen (2010) and Johnson and Klee (2007) can be applied to correlate the types of leadership styles identified by Bass (as shown in Turner & Müller 2005) with the Life Styles Inventory types (Lafferty 1973, in Cooke & Rouseau 1983) and the Dynamics of Group Interaction (Cooke & Szumal 1994). Autocratic leadership style can be correlated with power-aggressive behaviour (see Table 4) that suppresses constructive, and promotes passive, behaviours suggested by the dynamic of group interaction styles. The results reported by Berson and Linton (2003) support the correlation of transformational leadership style with LSI constructive behaviours that promote constructive behaviours, also suggested by the dynamics of group interaction styles. The transactional leadership style can be correlated with the competitive-aggressive or competence-aggressive LSI style (see Table 4), as a leadership style that is still aggressive but less aggressive than the autocratic.

2.4.7. Knowledge and Learning

The definition of terminology used in this section is first presented, followed by the review of the implications of knowledge in the development of software-intensive products. The section concludes with a review of the impact of group structure and learning behaviours on group performance.

Terminology

The definitions that follow, also included in the Glossary of this thesis, were adapted from the Dictionary of Psychology (Reber & Reber 2001) to fit into the context of this research.

Knowledge is the body of information possessed by a person or, by extension, by a group of persons or a culture. Knowledge is information placed into a context. Knowledge of a particular field possessed by a person is also referred to as *expertise*. *Skill* is the ability to successfully apply knowledge to execute a task and achieve an intended goal.

Explicit or articulated knowledge is acquired through training, formal education, writings, books and other forms of media (Dampney, Bush & Richards 2002). Explicit knowledge is transmitted in formal systematic language (Nonaka 1994) and thus can be documented and stored.

Tacit knowledge is not documented. It is widely held by individuals in the form of expertise, skills and know-how, although not readily expressed. Tacit knowledge is transferred through shared experiences and cooperation, and acquired through more intimate relationships such as with a teacher and apprentice (Dampney et al. 2002).

Learning is the process of acquiring knowledge. Although this definition is regarded to convey a loose meaning, it is sufficient for the purpose of this research. Learning is then defined as ‘the process of acquiring information and knowledge needed to perform a task’. Within this context learning can occur in many ways. Formal teacher-student training; practical hands-on-work training; and collaboration through workshops and mentoring; gathering of information from documentation; and trial-and-error are forms of learning found in engineering projects.

Experience is the knowledge gained by a person from participating in an event. Experience is acquired when a person puts into practice the knowledge he or she already has, allowing to develop mental models of events not yet experienced. Learning can therefore also occur through experience.

Five Orders of Ignorance

Software development can be viewed as a process of knowledge acquisition and ignorance reduction, and can be classified in five orders of ignorance, shown in Table 5 (Armour 2000, 2001, 2004).

Table 5 – Five Orders of Ignorance

Order of Ignorance	Meaning	Knowledge Available
Zero Order of Ignorance (0OI)	Lack of Ignorance	Most if not all knowledge required is available; and the kind of knowledge to be acquired is known and easily obtained.
First Order of Ignorance (1OI)	Lack of Knowledge	Some of the required knowledge is available, but not all; and the kind of knowledge to be acquired is known and can be acquired.
Second Order of Ignorance (2OI)	Lack of Awareness	Some of the required knowledge is available, but not all; the kind of knowledge to be acquired is unknown, but there is a suitable process for discovering in place.
Third Order of Ignorance (3OI)	Lack of Process	Some of the required knowledge is available, but not all; the kind of knowledge to be acquired is unknown, and there is no suitable process of discovering in place.
Fourth Order of Ignorance (4OI)	Meta Ignorance	There is no knowledge about the Five Orders of Ignorance.

Armour argues that each of the Orders of Ignorance requires a different type of process. Processes only allow things to be done that are already known, therefore well-defined and detailed processes are applicable to projects that are at Zero Order of Ignorance and to some degree and with fewer details to projects at the First and maybe Second Order of Ignorance. Software projects that are at Third Order of Ignorance require a completely different kind of process, that is, a process to discover ‘what we don’t know that we don’t know’, which is beyond the boundaries of the software development

process. Software development processes are limited to define how to implement existing knowledge into an executable software form.

For projects at the Third Order of Ignorance a learning-based process is required and the project would move to the Second Order of Ignorance. According to Armour, agile methods are indicated as a learning-based process for software projects and classified as 3OI and 2OI (Armour 2004 pp. 153-155). Agile methods are flexible and adjustable; they acknowledge that change is expected; they proceed in stepwise incremental development; and recognise the importance of human factors to address the knowledge-centric nature of software development (Armour 2004 pp. 104-106). ‘Agile is all about cooperative human endeavours, and the net purpose of these activities is the accumulation and proof of a large variety of knowledge’ (Armour 2004 p. 136).

Turner (2007) suggests that Systems Engineering is a form of learning-based process. Systems Engineering provides a set of tools and approaches that help to fill the gaps of knowledge that exist in the early phases of the project. Systems Engineering also helps to reveal the unknowns through collaborative activities such as requirements workshops and reviews, design evaluations, and demonstrations of prototypes and models that foster sharing the knowledge distributed among stakeholders and creating a shared collective knowledge. These activities can create knowledge through the emergent properties of the group. When knowledge is shared, people rethink what they knew and develop new concepts that, after validation and acceptance, become new knowledge.

Organisational Knowledge Creation

The Dynamic Theory of Organisational Knowledge Creation (Nonaka 1994) explains that the source of organisational knowledge creation is the individual. According to Nonaka, individuals share experiences and tacit knowledge through socialisation and externalise tacit knowledge into explicit knowledge. Individuals transform explicit knowledge into tacit knowledge through internalisation, a process that Nonaka indicates to be similar to the traditional notion of learning. The continual dialogue between tacit and explicit knowledge drives the creation of new ideas and concepts, and the process of combining and reorganising explicit knowledge can also lead to new knowledge (Nonaka 1994).

Knowledge sharing is a complex social process and social motivation theories better explain that while individuals receive knowledge from others they also transfer their

knowledge to others (Qinxuan & Yingting 2010). The authors suggest that managers should be aware of an individual's motivation factors and should encourage their employees to learn from each other.

Through knowledge sharing, personal knowledge can be transformed into organisational collective knowledge, and managers should find ways to remove obstacles and stimulate employees' behaviours towards knowledge sharing (Xio-qing & Nan, 2010). In consonance with Deming's System of Profound Knowledge Xio-qing and Nan refer to psychology theory to suggest that managers need to understand what motivates their employees, and recognise the influence of intrinsic and extrinsic motivators. In accordance with the Work Preference Inventory (WPI) intrinsic motivation is concerned with: self-determination, competence, task involvement, curiosity, enjoyment and interest; and extrinsic motivation is concerned with: recognition, competition, money and other tangible incentives (Loo in Xio-qing & Nan 2010). The authors conclude that the organisation can benefit from programs that stimulate knowledge sharing through motivating employees in accordance with their preferences.

Knowledge sharing is a complex social process and social motivation theories better explain that while individuals receive knowledge from others they also transfer their knowledge to others (Qinxuan & Yingting 2010). The authors suggest that managers should be aware of an individual's motivation factors, also in consonance with Xio-qing and Nan (2010), they should encourage their employees to learn from each other.

Team Structure and Learning Behaviours

As discussed in Section 2.1.4, the development of software-intensive products calls for specific domain knowledge to solve the problem, and knowledge of software technology to translate the solution into software code (Armour 2004). These two types of knowledge are necessary but not sufficient.

Individuals and even organisations in isolation will not possess all the knowledge. Part of the knowledge is available and distributed among the stakeholders. Knowledge distribution can be defined in terms of 'who has what type of information' (Rulke & Galaskiewicz 2000). Other required knowledge may be non-existent or not easily obtained. Cooperation and collaboration is therefore necessary to share and combine existing distributed knowledge into collective knowledge. As a consequence, knowing

how to work as a team is also necessary (Chan, Jiang & Klein 2008). The team members must be knowledgeable about learning-based processes, how the knowledge is distributed, and willing to collaborate and cooperate.

The performance of groups that have knowledge distributed across group members vary with the group structure, which is defined by the quality and patterns of relationship among the members of the group (Rulke & Galaskiewicz 2000). Hierarchical structures are more efficient for simple problems and tend to perform not so well for more complicated problems. Complex problems are better handled by decentralised groups of specialists that have close ties. Conversely it is expected that groups of specialists with less intense ties and an hierarchical structure will show poorer performance when dealing with difficult and complex problems.

Learning behaviours such as seeking feedback, sharing information, asking for help, talking about errors and experimenting, enhance team performance and team satisfaction (Yeh & Chou 2005). Teams that engage in these types of activities improve the learning process by sharing expertise and experiences among team members, discover unexpected consequences, and moderate the impact of team conflicts on team performance.

It can be argued that group structure, learning behaviours and knowledge of learning-based processes can be associated with the development of desirable emergent properties like quality and knowledge, as discussed in Section 2.4.6.

2.5. Social Systems of Engineering Projects

Lawson's research (2005) investigated the social aspects of engineering projects. The social system of an engineering project includes any group of people that has some association with the project or a stake in the outcome. Software-intensive projects are not dissimilar to engineering projects and Lawson's findings are pertinent to this research.

Lawson acknowledges that socio-technical systems are complex adaptive systems and as such the capabilities of the system will emerge only after the system is formed and in operation. Therefore, the predictive capabilities in constructing social technical systems cannot be claimed as properties of the system under construction. These are in fact properties of yet another complex adaptive socio-technical system, where individual and

collective experiences, explicit and tacit knowledge are applied to construct the target socio-technical system.

It is evident that in the process of engineering software-intensive systems several socio-technical systems are formed. The final outcome, i.e. a software-intensive solution to satisfy a need, is the compound result of multiple complex adaptive socio-technical systems, and therefore difficult to predict.

Lawson proposes the Social Systems Evaluation Framework (SSEF) to assess the project in three levels: social groups identified in the project; forms of capital in each group; and social factors underpinning the forms of capital.

Social groups encompass engineering teams; inter-linked teams; management interface; client interface; and societal interface. Forms of capital include embodied capital, social capital, institutionalised capital and economic capital.

Embodied capital is a property of the individual and reflects the contribution of each person in the group being tacit knowledge and experiences both technical and managerial; the capacity to learn and collaborate in the group; and emotional competence.

Social capital is a property of the community comprising fourteen indicators that include shared ways of doing things together; knowing what others know, what they can do, and how they can contribute to the enterprise; specific tools; the representation of artefacts; and, certain styles recognised as displaying membership of the group, among others.

Institutionalised capital and economic capital are both the property of the organisation. The first includes institutionalised processes for engineering; supervision and review of projects; reporting and accountability; realistic and regular review of estimations; mentoring and succession planning; and administrative tasks. Economic capital provides physical resources; technical infrastructure; and the funds to engage qualified and experienced people, purchase materials and to pay expenses.

The author indicates that although control and prediction of instrumental factors may be necessary for project success it may not be sufficient, and project management becomes increasingly focused on instrumental factors and less on social factors. The author

concludes that successful projects are those that also take social aspects into consideration, and are focused on the character and behaviour of the people involved in the project. Lawson's findings are consonant with Ropohl's philosophy of socio-technical systems and Deming's System of Profound Knowledge.

2.6. Summary

The review of the literature revealed the existence of a paradox about the perceived causes that drive success and failure of software-intensive projects, suggesting that there is a flaw in the paradigm in use. The quest for a better paradigm defines the objective of this research.

The System of Profound Knowledge was chosen to establish the path for the research: to appreciate the system of software-intensive acquisitions and understand its variations, in searching to formulate theories that would allow prediction and better conditions and to create and manage software-intensive acquisitions.

The review of systems and complexity theories indicated that software-intensive acquisitions have characteristics of socio-technical systems and complex adaptive systems. As such, software-intensive acquisitions operate on complex fitness landscapes and depend on emergent properties to be able to achieve the intended goals. Three desirable emergent properties have been identified: quality, knowledge and core competency. These properties cannot be planned nor predicted and the system should be managed at the edge of chaos in an attempt to steer the system to produce the desired outcomes without necessarily controlling it.

The development of software-intensive products requires not only knowledge about software technology, but also specific domain knowledge to solve the problem before the solution is translated into software code. Lack of knowledge is likely to propagate as distortions in the software product and the final *solution* may not satisfy the *need*.

Knowledge is distributed among stakeholders, and they must be knowledgeable of learning-based processes and teamwork to be able to share existing knowledge to create collective knowledge. Group structure and learning behaviours are factors that influence the development of knowledge as a desired emergent property.

People can cause the enterprise to succeed or to fail. Knowledge of psychology and social behaviour is necessary not only to understand the system but also to motivate people to do what is needed to succeed. An investigation of the social system of engineering projects was also reviewed and confirmed that social aspects are to be taken into account when planning and managing engineering enterprises, in which software-intensive acquisitions are included.

The review of the literature revealed three clues that will guide the quest for a better paradigm for the causes of success and failure of software-intensive projects and acquisitions. We need to understand the acquisition as a socio-technical system; the influence of availability and absence of required knowledge; and the variations in the system. Improving understanding in these areas will reduce uncertainty and will contribute to the creation of a framework that will allow better prediction of the chances of success and failure of software-intensive acquisitions.

Chapter 3: Appreciate the Software-Intensive Acquisition System

This chapter addresses the first part of the System of Profound Knowledge: appreciate the system. It will present a narrative about the nature of the software-intensive acquisition system, elaborating what was discovered from the review of the literature. The focus will be on following the first two clues suggesting that there is needed a better understanding of the socio-technical system and the importance of knowledge in software-intensive acquisitions. The system will be discussed conceptually, in an attempt to capture the characteristics that are common to acquisitions that require the development of software-intensive products. The conclusion of the narrative will lead to the formulation of hypotheses that after testing and validation in a simulated environment will contribute to the creation of a framework to improve prediction in software-intensive acquisitions.

3.1. System in Context

The intended and explicit aim of the acquisition system is to devise a solution that will satisfy a need. Software-intensive acquisitions, in particular, require solutions that rely on engineering projects to develop products with a strong dependency on software components.

Software-intensive acquisitions are socio-technical systems (discussed in Sections 2.3 and 2.5). The technical portion of the system comprises technology used to develop and implement the subject of the acquisition that is also a technical system, a collection of hardware, software and operational procedures that are intended to be a solution for a need. The technical system does not have the capacity to change itself and is created and modified by the social component of the system. The social system, comprising people associated in teams and organisations, is responsible for realising the technical system and has the capacity to change itself as well as changing the technical system.

The aim of the social system is to realise the technical system and achieve the objective of the software-intensive acquisition. Systems do not always have an explicit aim, but every system has an implicit aim, being what the system does (discussed in Section 2.3.1) as the result of the interaction of its parts. Organisations, however, have explicit

and implicit aims and need to be managed to be able to achieve the intended purpose of the system. The explicit aim of the social system of software-intensive acquisitions is to acquire a solution that depends on software products to satisfy a need.

The acquisition also has implicit aims, whether economic, political or personal. Implicit aims interfere with explicit aims in good and bad ways. Understanding how the system is influenced by explicit and implicit aims is one of the aspects of ‘appreciating the system’ of software-intensive acquisitions, discussed in Section 2.2.1.

Leadership and management are what guide social systems to achieve their intended purpose, discussed during the review of the literature in Chapter 2. The intended purpose of the system has to be communicated and the components of the system (i.e. people, teams and organisations) need to be motivated to do what they have to do to achieve the intended purpose as an emergent property of the system (see Section 2.4.5), which is a task for leadership and management.

3.2. Capability Development and Acquisition Processes

In Australia, the Defence Capability Development Manual (DCDM) (DOD-AU 2006) defines the process for defence capability development. Capability development comprises two basic phases: Need identification and Requirements specification. The Need phase identifies a capability gap that needs to be addressed. After the Government endorses the need, the Requirements phase specifies a well-defined and costed solution to that need, which after approval is budgeted and scheduled for acquisition.

The Australian Standard Defence Contracting policy (ASDEFCON – Strategic Materiel) (DOD-AU 2009b) defines the acquisition process of major defence systems, named Strategic Materiel. The ASDEFCON is a tendering and contracting template and specifies that the contractor shall conduct its program of engineering activities in accordance with the requirements of EIA-632 ‘Processes for Engineering a System’ (EIA 1999) the standard that adopts the Systems Engineering approach (see Section 3.2.2). The procurement of defence materiel in Australia follows the Defence Procurement Policy Manual (DPPM) (DOD-AU 2010) for the acquisition tendering and contracting processes of software-intensive systems for defence.

The success of the acquisition thus depends upon how well the Capability Development and Acquisition Processes are executed.

3.2.1. Life Cycle Models

Life cycle models determine the sequence of activities for acquiring and developing systems and software. For the purpose of this research the development life cycle starts at the specification of requirements to satisfy the need, and ends when the solution is put into operation.

Software development life cycle defines the sequence of software development activities: conception, specification, design, implementation, review, integration, test, and maintenance of software components (McConnell 1996). The system development life cycle has a wider scope and establishes activities to transform the needs of stakeholders into the products that will satisfy that need through the specification, design, implementation, review, integration, test, maintenance and disposal of the system (ISO/IEC STD-15288 2008). The components of the system may include hardware, software, and procedures related to the operation, training, safety, security, and anything else that is part of the system that will be delivered to the customer.

There is a strong link between the system and its software as software is an integral part performing essential functions in the system (ISO/IEC STD-12207 2008). There also a strong link between software and the systems engineering life cycle for without the correct definition of the needs of stakeholders and the specification of system requirements there is no guarantee that the software will perform its functions.

Many life cycle models exist, each with its own characteristics that can be more or less suitable to the characteristics of the project. Three types of model will be considered: sequential, incremental and evolutionary. The characteristics and suitability of each model that follows have been compiled from McConnell (1996) and Stevens et al. (1998).

Sequential Model

The *sequential* or *waterfall model* is characterised by a sequence of activities executed in one pass, each interacting only with its previous and subsequent activity. The sequential model is the most well-known model and the first used to build complex systems. It has been proven to work well for complex projects that are well understood and without risks, and in these conditions its well-defined structure can compensate when there is a shortage of experienced staff. It does not work well for projects that are

not fully understood from the beginning because it is difficult to complete the specification of requirements at the beginning of the project. As well, the illusion of prediction from early unrealistic plans imposes a serious risk to the project. There are variations of the sequential model (e.g. waterfall model with feedback; with overlapping phases; with sub-projects; and with risk reduction) that attempt to compensate for some of its deficiencies. However, as software-intensive projects become more complex, the waterfall model becomes less attractive.

Incremental Model

The *incremental model* follows the sequential approach to conceive, specify and design the system and is then followed by implementation in incremental parts. This model works well when the sequential portion of the life cycle produces a good specification and design. As for the sequential model, the incremental approach produces good results when the project is well understood and the implementation of incremental phases can be well defined. The application of the incremental model to complex projects is therefore limited. An advantage of the incremental model over the sequential model is the shorter delivery time, allowing the customer to see and maybe use part of the system earlier and raise warning flags when the product does not meet expectations. Because the model does not recognise that requirements may change as a consequence of the development process and customer feedback, the incremental model like the sequential model, still creates changes that may result in cost and schedule variations. The *evolutionary model* addresses this situation.

Evolutionary Model

The *evolutionary model* is indicated for complex projects that cannot be developed by sequential and incremental models. The model recognises that not all is known about the system under development and the process of discovery occurs during the development cycle. The basic development life cycle is repeated to deliver successive increments of the system, which after being put into operation provide the feedback to define the next evolutionary increment. Acquisition of knowledge is a continuous task and users, customers and developers cooperate and learn about the system that should be built to meet the needs of the user. The next phase is planned, if one is needed, and the requirements of the system are specified as the system evolves and more becomes

known about it. Evolutionary increments should be chosen to maximise value, starting from infrastructure and then adding functional modules.

A reported disadvantage of the evolutionary model is that it is complicated and requires continuous and attentive knowledge management so that it does not get out of control, which should be expected considering the ‘Law of Requisite Variety’, discussed in Section 2.3.1. Only variety can absorb variety, i.e. the system can be controlled only by a system as complex as the one under control. Another critical and difficult aspect of the evolutionary model is to determine what should be the size and frequency of each evolutionary increment: if the increment is too small and frequently applied the process may become inefficient; if the increment is too large and application spread, the process may become out of control.

The spiral model for software development (Boehm 1988) is an evolutionary model and is indicated for the development of large software-intensive systems²⁴ (Boehm, Brown & Turner 2005).

3.2.2. The Role of Systems Engineering

Systems Engineering (SE) is a disciplined approach to engineer technical systems; and a learning-based process, discussed in the literature review (see Section 2.4.7). SE can be applied following sequential, incremental and evolutionary life cycle models (Bar-Yam 2003; Gilb 2005; Stevens et al. 1998).

The International Council for Systems Engineering (INCOSE) defines as follows:

Systems Engineering is an interdisciplinary approach and means to enable the realization of successful systems. It focuses on defining customer needs and required functionality early in the development cycle, documenting requirements, then proceeding with design synthesis and system validation while considering the complete problem: operations, performance, test, manufacturing, cost and schedule, training and support and disposal.
(INCOSE 2004)

²⁴ The Win-Win Spiral Model (Boehm, Brown & Turner 2005), a variation of Spiral model with even more emphasis in stakeholder participation, is used on probably the largest and most transformational systems under development today: the U.S. Army/Defense Advanced Research Projects Agency Future Combat Systems Program.

The definition above suggests that systems engineering could be the answer sought to succeed in complex software-intensive projects, a view also expressed by a recent audit report on the acceptance of Australian Navy capabilities (ANAO 2010a). It has been demonstrated that SE effort has a positive effect on cost and schedule compliance (Honour 2004). The GAO 2009 report on Defence Acquisitions of Selected Weapon Programs (GAO 2009) acknowledges that systems engineering activities applied at the early stages of the acquisition help to acquire the knowledge about the need and the technologies to be used in developing the solution, as well as the knowledge to estimate cost and schedule. Adequate knowledge at the right time allows a more realistic and accurate estimation, reduces changes in requirements and consequent rework at late stages of the project, thus reducing cost and schedule variation against the plan.

The ROI of systems engineering is maximised when the right amount of SE activities is applied. Larger and more complex systems require more SE effort than smaller and simpler projects (Boehm, Valerdi & Honour 2008). Honour (2004) suggests that to estimate the optimum SE effort one must take into consideration a number of basic values, some well understood by management (cost, duration and risk) and some very intuitive values associated with the technical product (size, complexity, quality and the overall value of the technical product). If the SE effort is too little the technical product may not reach the value sought, while SE effort beyond the optimum point would increase costs without improving the final product resulting in a poor ROI. The optimum SE effort suggested by Honour is in the order of 15% to 20% of the total effort for the project.

Reasons why Systems Engineering may not be applied

If the effectiveness of SE has been demonstrated and the ROI of SE activities can be maximised, why is SE not consistently applied? SE is part of what is known that helps software-intensive projects to succeed, and part of the flaw in the paradigm that creates Cobb's Paradox (see Section 2.1.2). The explanation comes from the fact that the ROI of SE and the value added to the final product are not unquestionably agreed and the reduction of SE activities is often a cost-cutting measure (Boehm et al. 2008). SE is not consistently applied because it may conflict with other management objectives, confirming that managers prefer to take poorly managed risks in the hope to obtain short-term results (see Sections 1.2, 2.2 & 2.3.2).

3.3. The Conceptual Acquisition Process

The aim of the software-intensive acquisition is to engineer and produce a software-intensive solution that will satisfy the need. The solution depends on how well the need was identified and communicated and how well the solution is specified and communicated to the organisation that will engineer and develop the software-intensive system to address the capability gap. The conceptual process thus includes capability development and acquisition processes.

The solution comprises ‘end-products’ and ‘enabling-products’, introduced while discussing the system’s effectiveness and efficiency in Section 2.3.1. End or operational products are the elements that will be delivered to the ultimate user and that will be applied as the solution that meets the need, being hardware, software, facilities, people and services. The processes that create operational products use and also create other enabling products, where enabling products include artefacts of development, test, training, production and support. As a system, components of enabling and operational products are inter-related and complement each other so that the operational products will perform in a way to satisfy the original need of the acquisition. ‘Enabling-products’ are of value during the development of the operational products and are discarded when the latter enter into operation.

The quality of the solution is determined by how well the ‘end-products’ meet the need when put into operation. The total cost and time to produce the solution is the sum of cost and time to produce all enabling and operational products.

Products, whether operational or enabling, can be physical or functional in nature. Physical products have physical properties, while functional products are usually ideas, concepts and specifications of products that do not exist physically (Aslaksen 1996). Software itself does not have physical properties and is a functional product. A functional product must be instantiated into a physical product to exist. The software design is instantiated in the software code that determines the operation of a hardware component. Products can be inputs to tasks to produce other products and both physical and functional products are the result of tasks executed by people.

The software-intensive acquisition model will therefore include a representation of operational and enabling products; the tasks to produce these products; and the people that execute the tasks.

3.3.1. Transformations through Domain Spaces

The software-intensive acquisition process is a form of collaborative design. Collaborative design is the process of developing physical and behavioural products²⁵ that can be represented in a multi-dimensional space (Klein et al. 2001). Products are represented as a set of parameters with a unique value captured in the form of requirements, specifications and processes for creating artefacts. Each parameter can be represented by a value placed along its own orthogonal axis of a multi-dimensional space referred as the design space. Every point in the space corresponds to a distinct solution, although it may not be the best or feasible. The aim of the collaborative design is searching for a point in the design space that corresponds to a solution within acceptable constraints. Multiple participants whether individuals, teams and organisations perform the process of developing artefacts produced by collaborative design.

Software-intensive acquisitions are a form of collaborative design. Therefore, the value of each attribute that defines the need and solution in the software-intensive acquisition process can be placed along its own orthogonal axis of a multi-dimensional software-intensive acquisition space. Each point in the software-intensive acquisition space will lead to a distinct solution that may or may not satisfy the need or meet the imposed constraints. The aim of the software-intensive acquisition is searching for a point in the software-intensive acquisition space that corresponds to a solution that meets the explicit aim of the acquisition. The ability of the software-intensive acquisition system to meet the explicit aim is determined by the capacity of the people that will search in the acquisition space for an acceptable solution. It can be asserted that if the people do not have the capacity in the form of knowledge, skills and experience required to perform the task, the solution may not be found within the constraints of cost and schedule.

²⁵ Physical products can be equipment such as aircrafts, ships, land vehicles or a computer system; behavioural (also referred as functional) products can be specifications, plans, schedules, processes or software.

Products are a collection of artefacts associated with the domain space in which they exist. Products and their artefacts are described by functional and performance attributes. The acquisition space contains all operational and enabling-products. Products can be grouped logically in accordance with sub-domains of the acquisition space. Three main sub-domain spaces within the acquisition space are defined: Situation Domain, Problem Domain and Solution Domain.

The acquisition originates in the Situation Domain, where a need exists and can be expressed by a set of products in the Situation Domain. The need leads to the definition of the problem as another set of products in the Problem Domain. The solution that resolves the problem comprises a set of products in the Solution Domain. The transformations of products from need to problem and then to solution require human resources with knowledge and skills specific of each domain.

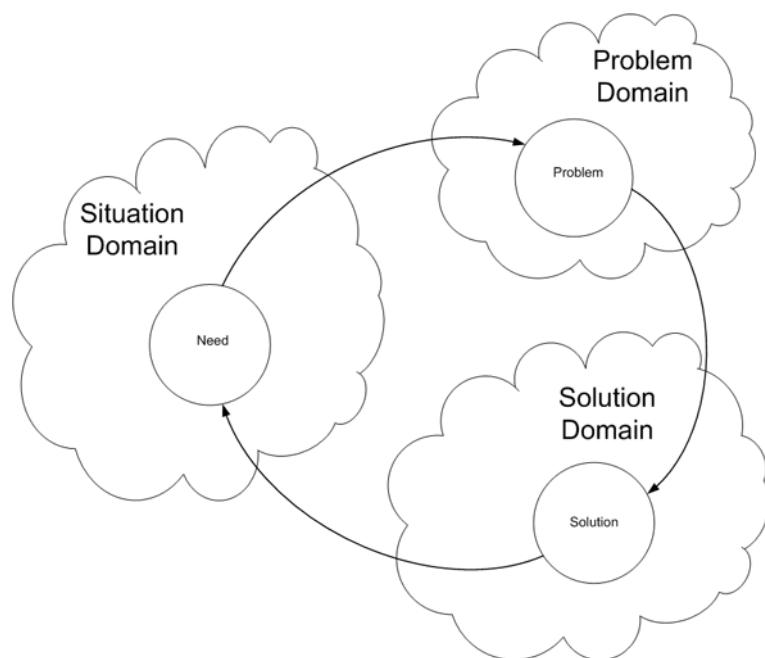


Figure 1 – Transformations between Domain Spaces

Both the situation and solution domains contain physical and functional products. The need is perceived and expressed in the form of specifications of functional products. The solution is defined also as functional products, such as specification and design, and implemented by physical products, such as equipment and operational procedures. The problem domain is functional in nature. Ideally, the process should occur as shown in Figure 1, where the need is accurately expressed, the problem is well defined, the solution is correctly implemented and satisfactorily addresses the need.

The execution of the tasks that produce the products requires knowledge associated with the domain space of the input and output products. The quality of the task execution will depend on people's knowledge and skills and their motivation to execute the task.

Figure 2 shows the real case. Within the Situation Domain, divergence occurs as the 'Real Need' is interpreted into the 'Perceived Need'. As the latter becomes the 'Expressed Need' it incurs further distortion. Thus, within the Problem Domain, the derived 'Expressed Problem' carries the sum of all these distortions to the Solution Domain. The resulting 'Solution Implemented' is quite unlikely to satisfy the Real Need.

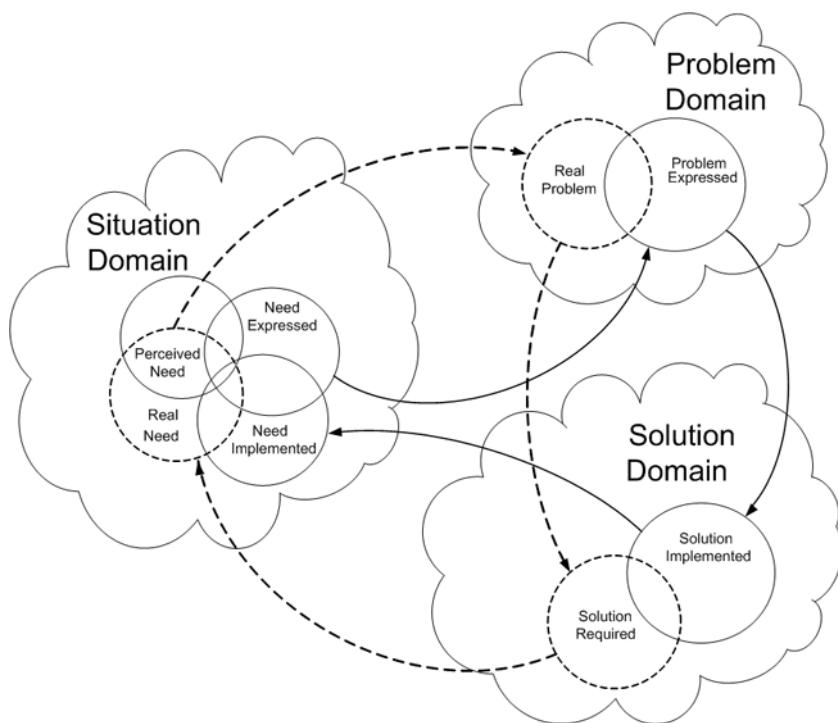


Figure 2 – Actual transformations between Domain Spaces

Distortions occur because people in the social system, limited by their own knowledge, skills, cognition, emotions, moved by personal and corporate goals and bounded by their roles in the organisation, are responsible for the execution of transformations between and within domains.

The ideal transformations should accurately move from one domain to another without distortions, leading to a solution that satisfactorily improves the situation. Success thus depends on how well the transformations occur.

3.3.2. Products and Tasks in the Acquisition Space

Products are documents, hardware and software developed during the acquisition. Tasks are the activities that transform products into other products. From communicating the need to the end-product delivered to the customer, the acquisition is a series of tasks that transform products. Software development is part of these transformations.

The need is expressed through a set of documents describing the concept of operation and a functional performance; these documents are transformed into other documents defining the solution in the form of system specification and design. When the solution requires software products the system documents are transformed into software requirements and then into software design; and these are finally transformed into software code. Support products such as plans, reviews and testing are also results of transformation tasks.

The solution is likely to include products realised by hardware, software and operational procedures. The development of software, like any physical product, requires specification, design and construction. As the interest of this research is in the implications of the development of software, the model proposed in Chapter 5 will focus on software products and their respective enablers.

Effort, Knowledge and Complexity

Transformation tasks require a level of effort and knowledge to be correctly executed. Simple tasks can be well defined and can be well understood. Complicated tasks can be well defined but may be difficult to understand due to intricacies of details. Complex tasks cannot be fully defined and bear unknowns that have to be discovered and resolved so that the task can be completed successfully. The process of discovering requires effort that is not always planned.

Transformation tasks that collaborate to the end-product apply engineering processes and require knowledge of input and output products. A software engineer should be able to understand the system requirements to be able to write the software requirement's specification document (SRS); while a software developer should be able to understand the SRS to write the software design document (SDD); and the next software developer should be able to understand the SDD to write the code. Similarly any other transformation in the process of development of a software-intensive solution requires

knowledge of the input domain and of the output domain. Lack of knowledge and skills in the input and output domains of a transformation task will lead to distortions and omissions in output product.

Error Propagation

The acquisition process is a sequence of transformations where distortions, errors and omissions that occur in the previous transformations are carried to the next transformation as in a *Chinese Whispers*²⁶ effect. Specification and design of software components depend on the completeness and quality of documents that are a result of previous transformations. Distortions in previous transformations are carried into the software specification, design and into the code. To correct these defects requires information and knowledge from previous domains that are not always available in the software domain. Lack of information, knowledge and expertise of capabilities and products identified in previous domains while the software is in production is one of the most common reasons for software delays and cost increases, although not always recognised.

Among the causes of software project failure are inaccurate estimation and changing requirements (see Section 2.1). Lack of knowledge and skills are drivers of poor specification and estimation and can be associated with the root causes of software project failure. Lack of knowledge and skills are often present in engineering and in management: lack of engineering knowledge and skills to find a solution before the software development starts (see Section 2.4.7); and lack of management knowledge and skills to recognise that there is a deficiency in engineering capability and adequately estimate, plan and manage this deficiency.

²⁶ *Chinese Whispers* is used as a metaphor for cumulative error when information is transmitted from one recipient to another. The metaphor is based on a game Chinese Whispers, also known as Telephone, Grapevine, Broken Telephone and Whisper Down the Lane, where the first player whispers a phrase or sentence to the next player. Each player successively whispers what that player believes he or she heard to the next. The last player announces the statement to the entire group. Errors typically accumulate in the retellings, so the statement announced by the last player differs significantly, and often amusingly, from the one uttered by the first (source Wikipedia, http://en.wikipedia.org/wiki/Chinese_whispers).

3.3.3. Effectiveness, Efficiency and Performance

Effectiveness indicates how well the acquisition achieves its goals and *efficiency* represents how much waste is produced in the process. Effectiveness will be assessed comparing the real system against the ideal, as discussed in Section 2.3.1. From Figure 2, effectiveness indicates how close the implemented solution came to fulfil the real need, i.e. the unbroken line circles coinciding with the dotted circles, and efficiency shows how much effort spent on support tasks is required to bring the implemented solution to meet the real need.

The efficiency of the system is determined by the ratio of productive and support tasks. Management is a support task that does not add value to the product delivered to the customer. However, the acquisition needs to be managed, and management is therefore a necessary task that reduces efficiency. Reviews and quality assurance activities are also support tasks that add value to the process and may indirectly add value to the end-product through the corrective actions that result from reviews. The effort spent on reviews that reveal no errors decreases efficiency; when errors are found and corrected the rework task will add value, increasing cost and schedule. Management and reviews are planned tasks to make sure the acquisition does what it is supposed to do. Both will lower efficiency and contribute to the system achieving the target effectiveness. Therefore, the acquisition will have a planned nominal efficiency to be effective, also discussed in Section 2.3.1.

Performance is an indication of how well the system does what it is supposed to do. As discussed in Section 2.3.1, performance can be subjective and there is no acceptable definition of performance that could be applied to different processes.

Performance for the purpose of this research will be used to compare the *real system* against the *ideal system*. The *ideal system* will always achieve the *Ideal Effectiveness* (E_I), at *Ideal Cost* (C_I), taking *Ideal Duration* (D_I) at *Nominal Ideal Efficiency* (N_I). The *real system* will achieve its *Nominal Effectiveness* (E), at *Nominal Cost* (C), taking the *Nominal Duration* (D) at *Nominal Efficiency* (N). *Effectiveness* and *Efficiency* are dimensionless numbers between 0.0 and 1.0.

The *Cost Performance* (P_C) and *Schedule Performance* (P_S) are defined as:

$$(3.1) P_C = \frac{C_I}{C} \quad (3.2) P_S = \frac{D_I}{D}$$

The *Effectiveness (E)*, P_C and P_S of the *ideal system* are *1.0*. Real systems would have E , P_C and P_S of less than *1.0* if they produce less than the ideal solution and their costs are higher and the schedules are longer than the ideal system.

In the ideal system people possess knowledge and skills required to execute the transformation tasks, while in the real system this condition will be less than ideal. Lack of knowledge and skills, or lack of application knowledge and skills available, are some of the causes of low effectiveness in software-intensive acquisitions, and the need for rework will increase the time and effort needed to do an acceptable job. Rework adds value to the end-product and does not lower the efficiency, but it may reduce performance by increasing cost and extending schedule, discussed further in this section.

3.4. Knowledge and Learning in Software-Intensive Acquisitions

The importance of knowledge and unrealistic expectations about foreknowledge in software-intensive acquisitions were introduced in the literature review (Section 2.1.4). This section further discusses how management and technical knowledge impact software-intensive acquisitions.

3.4.1. Management Knowledge

The theory of knowledge was introduced as part of the System of Profound Knowledge in Section 2.2.5, where knowledge is the result of theories created to predict future behaviour of the system. In that context knowledge conveys the understanding of the system from an outside view. Managers and decision makers, although being part of the system, need to understand the system from the outside to be able to design and manage the software-intensive acquisition system.

It is all about Knowledge

The design of the acquisition should address the knowledge that is needed, the knowledge that is available and be aware of the gap in knowledge and skills still required to engineer the software-intensive solution. The acquisition design should also

take into consideration how knowledge is distributed among stakeholders and plan to create collective knowledge and acquire what is not available.

In the development of complex software-intensive solutions there are unknowns that are known, and unknowns that are unknown. For a realistic assessment of knowns and unknowns the acquisition should be classified in accordance with the ‘Five Orders of Ignorance’ (see Section 2.4.7 and Table 5). Simple acquisitions would be classified as 0OI or 1OI and complex acquisitions with good chances of success should be classified as 2OI. If the classification indicates 3OI, a process of discovery should be put in place and the acquisition would become 2OI. Evolutionary development implements a process of discovery and it is an approach to move acquisitions from 3OI to 2OI. The acquisition would be destined to failure if it is classified as 4OI or 3OI.

Unrealistic expectations about foreknowledge (Turner 2007 p. 12) discussed in Section 2.1.4, can increase costs and extend schedules and are one of the root causes of failure of software-intensive acquisitions. This situation can be explained when management is predominantly transactional. Under the pressure to meet cost and schedule task-oriented managers may not recognise that the acquisition is suffering from lack of knowledge and take measures that do not address the real problem and worsen the situation.

The Challenge of Estimating

Estimation is another important aspect in the realm of management knowledge. Every task requires knowledge (in fact any combination of data, information, knowledge and skills) and effort. If any of these two attributes are under estimated or do not conform to the ability of who (person, team or organisations) will perform the task the product of the task will contain distortions, whether because the task is not complete (effort underestimated) or will contain errors (data, information, knowledge or skills underestimated). Therefore managers need knowledge of how to estimate what a task entails and what is the capacity of the workforce that will perform the task.

Estimation implies a judgement, opinion or approximate calculation and therefore carries a risk of being wrong. Estimation of software development or complex engineering systems that has never been done before is particularly risky and often is wrong. This is not only because it is difficult, if possible at all, to estimate a task that has not been done before, but also the abilities of the person that will execute the task

are not known. Therefore, the ability of the estimator to estimate the task has to match the ability of the person that will do the task.

If the effort of the task is under-estimated but it is well defined and simple, i.e. 0OI or 1OI (see Section and 2.4.7, Table 5), performance as a function of effort applied is likely to be linear and the problem can be corrected by adding resources to the task. It can be hypothesised that if the task is 2OI or higher, the effort to acquire the required knowledge and discover unknowns makes performance non-linear. In this case, adding resources will not resolve and may even worsen the situation. This reasoning is the essence to the ‘Mythical Man-Month’ (Brooks 1995), and one of the biggest challenges of software development management.

3.4.2. Technical Knowledge

Software-intensive acquisitions require the application of specific knowledge, including knowledge of the field where a need was originated (domain knowledge) and knowledge of the software technology used to create the solution. As discussed in Section 2.4.7, unrealistic expectations are created when managers forget that only after the solution is defined software can be created, for software encapsulates a solution that often addresses a need in a domain foreign to software developers. The process of acquiring knowledge takes time and effort and it can be equated to cost and schedule. Therefore, underestimating the difference between the available knowledge and experience and what is needed to execute the task can be equated to cost overrun and schedule delays. Lack of knowledge, experience and skills is thus another factor that causes projects to underperform.

One aspect that makes software-intensive projects more difficult and susceptible to failure than other engineering projects that do not depend on innovative software, is the level of abstraction of requirements. The abstraction adopted to manage complexity may hide important details that propagate to the specification and design of software components (Bar-Yam 2003). This situation is characteristic of the complexity of software-intensive solutions and the diversity of domains of knowledge embedded in the software. To resolve these details requires domain knowledge often not available in later phases of the project which software developers would not be able to resolve. These are defects that reduce quality, increase costs and schedule.

Distributed Knowledge

No single individual or organisation possesses all knowledge, nor has access to all information required to specify, design and implement complex software-intensive solutions. Individuals and organisations are likely to be more knowledgeable about subjects that are pertinent to the domain where they operate. Distributed knowledge often leads to incomplete information and creates different perceptions and mental models. Distributed and unshared knowledge turn teams and organisations away from common understanding, common goals, and affects the system effectiveness and efficiency.

The effects of incomplete information and distributed knowledge combined with less than ideal skills create distortions that propagate through specification and design documents, resulting in incomplete or even wrong information when they reach the software domain. Problems that are not addressed and resolved in their domain of origin will propagate through the acquisition space to other subspaces where the required knowledge to address the problems is inadequate or non-existent. These distortions become defects that to be corrected increase cost and schedule, and become drivers of management intervention.

The larger the number of organisations in the system, the greater these effects of distributed knowledge become. The acquisition process, and consequently the software-intensive project, ought to impel the acquisition organisation, as a collection of several organisations, to harvest and apply collective knowledge. Collective knowledge is an emergent property, whose effectiveness should be maximised by structures that favour learning behaviours, collaboration and knowledge sharing. The next section discusses how this is attained.

3.4.3. Learning Processes and Behaviours

The uncertainty and the distributed nature of knowledge in software-intensive acquisitions is a fact that must be managed. Complex acquisitions must be placed at no higher than second Order of Ignorance (2OI), i.e. some of the required knowledge is available, but not all; the kind of knowledge to be acquired is unknown, but there is a suitable process of discovery in place. The process of discovery can include the process of Systems Engineering as a learning-based process (see Sections 2.4.7 and 3.2.2) and

an evolutionary development life cycle as the process of discovery and uncertainty reduction that evolves and mitigates risks.

Technical and management knowledge are essential but not sufficient and teamwork skills are also necessary, as discussed in Section 2.4.7, to facilitate knowledge sharing, collaboration and creation of collective knowledge.

Dynamic of Group Interaction and Software-intensive Acquisitions

The acquisition design should also take into account the influence of group interaction styles in problem-solving groups (Cooke & Szumal 1994), discussed in Section 2.4.6. Constructive behaviours are likely to produce better results in the form of quality and acceptance of the solution, while aggressive and passive behaviours will lower both the quality and acceptance of the solution. Constructive behaviour is likely to facilitate collaboration and learning behaviours (asking for help; offer help; sharing information and knowledge; talking about deficiencies and defects, as discussed in Section 2.4.7) that will foster the creation of new and collective knowledge. Managers should promote a constructive attitude in the group, whether the group is performing well and particularly when the results are not as planned, and avoid aggressive behaviour.

Group members with more power, e.g. in management or leadership roles, or with stronger personality styles are likely to influence other members in the group (Cooke & Szumal 1994).

The Dynamic of Group Interaction (Cooke & Szumal 1994) explains that members in the group are likely to be influenced by group members with more power, and that aggressive behaviour causes a passive response. Therefore, staff will respond to aggressive behaviour coming from management with passive behaviour. This likely behaviour is also explained as a natural response to preserve their job security (safety: second level of ‘Hierarchy of Needs’, Maslow 1943), which will result in products with more errors that will take longer, and cost more, to fix. This situation is likely to cause a positive feedback: managers will become even more aggressive and the staff even more passive, further worsening the situation.

The undesirable aggressive-passive cycle can be broken by the drastic realisation that the task is impossible to be achieved, or by adopting constructive behaviours: the first results in total failure with the cancellation of the project; the second is not easy to

achieve. Troubled projects have a tendency to reach a stable aggressive-passive state, which counter-intuitively is a state of lower energy situated on vales or lower hills of the project's fitness landscapes (see Section 2.3.3). Managers assume that by exerting their power and adopting an aggressive behaviour, the staff will obey their commands and execute the project as planned, and the objectives will be achieved. Managers assume a command and control attitude of 'set-and-go'. Aggressive behaviour may require some personal energy to command, but after that it takes a low energy path of 'wait-and-see', at least until the next problem is encountered and another command is issued. The staff, on the other hand, that may have had their attempts at constructive initiatives suppressed by a manager, assume passive behaviour, which is also a path that requires lower energy by taking away from them the responsibility of doing what needs to be done to succeed and placing the responsibility onto the management by assuming the position of doing what they were told as in: 'management knows best, so ... whatever'. The developers provide the variety that is required to engineer the solution. Without their contribution, cooperation and full support, failure is imminent.

Changing to constructive behaviour corresponds to moving the project to higher peaks of the fitness landscape and requires energy. This energy comes in the form of personal effort to cooperate with members of the group and finding alternatives to reach the solution, and to fund the required resources. The need to apply personal effort is opposed by members of the group that have a tendency to behave aggressively or passively, while the expenditure of funds and likely schedule delays is likely to be opposed by management. The change to constructive behaviour has a better chance of success when it comes from management, because aggressive managers can easily suppress constructive initiatives initiated by the staff. The Dynamic of Group Interaction (Cooke & Szumal 1994) explains that when a constructive attitude comes from management and leadership, i.e. from group members with more power, the interaction style is likely to become constructive because the other members of the group tend to respond constructively.

3.5. The Software-Intensive Acquisition as CAS

Software-intensive acquisitions are socio-technical systems. The technical portion of the system is the subject of the acquisition as a collection of hardware, software and operational procedures that are intended to be the solution for a need. The technical

system does not have the capacity to change itself and is created and modified by the social component of the system. The social system, comprising people associated in teams and organisations, is responsible for realising the technical system, and has the capacity to change itself as well as changing the technical system.

The literature review, Section 2.3.5, introduced the concept of complex adaptive systems (CAS) and suggested that complex software-intensive acquisitions are CAS operating in a complex fitness landscape. Table 6 maps the software-intensive acquisition system against the seven basic characteristics of CAS (Holland 1995) and confirms that software-intensive acquisitions are CAS where people are the adaptive agents. The products acquired and developed by the acquisition can be considered as non-adaptive agents that are created and modified by adaptive agents.

Table 6 – Software-Intensive Acquisitions Seven Basics Characteristics of CAS

Properties (p) and Mechanisms (m)	Software-Intensive Acquisition System
Aggregation (p)	Organisations, groups, departments and teams aggregate people to perform purposeful tasks to meet contractual obligations and achieve a common objective.
Tagging (m)	People's formal roles and responsibilities together with informal perceptions of personalities and professional abilities create the identification mechanism that people use to relate and interact with others in the system.
Internal Models (m)	People are the adaptive agents in the system and their internal models are constructed based on their professional training, experiences and beliefs as well as the processes and procedures adopted in the system.
Building Blocks (m)	People's skills and experience in engineering, management and support activities provide the building blocks to construct new artefacts, manage the acquisition to meet explicit goals and deal with implicit, new and unexpected situations.
Non-Linearity (p)	The interactions of adaptive agents cause non-linearity. Adaptation changes the agent's 'internal models' which in turn changes their productivity. Non-linearity is also caused by the time spent on non-productive or poorly planned tasks such as learning how to do the work and the communication between adaptive agents.
Diversity (p)	Adaptive agents are the source of diversity that provides the 'requisite variety' for the acquisition to achieve its objectives.
Flows (p)	Explicit artefacts in the form of documents and products as well as ideas, explicit and tacit knowledge, rewards and reprimands, feelings and perceptions transferred between adaptive agents through formal and informal interaction, verbal and written communication or sensing.

3.5.1. Adaptive Agents

Adaptive agents in CAS are driven by their internal models that determine the agent's behaviour in response to how the agent perceives the environment and interacts with other agents. Adaptive agents are capable of learning from experience and in this way change their internal models to adapt to new situations. The concept of the adaptive agent model introduced in Section 2.3.5 includes a performance system based on simple rules and a process of changing the performance systems based on assigning credit to existing rules and discovering new rules.

Performance System

The performance system is a set of rules that the agent uses to assess the environment, process the interaction with other agents and determine the actions in response to the situation. The performance system thus represents the agent's capabilities and is specific for different kinds of agents. Agents that are not adaptive maintain the same performance system throughout their lives.

Complex performance systems can be constructed with rules of the form of 'IF < > THEN < >' statements, which process messages received from the agent's sensors, and may contain a very large number of rules organised into useful building blocks.

Human beings in social systems hold the extreme case of complexity of performance systems and adaptation. Rules in the performance system of human agents can be generic or very specific. Generic rules, although not guaranteed to lead to successful results are capable of handling new situations for which the agent may not have a specific rule. On the other hand, specific rules will be successful only in the same specific situation.

Take for example the set of rules shown in Table 7. These rules could be included in the performance system of an engineer agent in a social system of an engineering project and vary from a very generic Rule #1, to a very specific Rule #4. The first rule is applicable to any kind of problem and could be present in the performance system of any kind of intelligent agent. However the action that follows the rule will depend on further analysis and on the capacity of the agent. The analysis that should follow would involve other rules, as for example Rule #5. If the project is for example to develop medical equipment, the problem presented can well be in the field of medicine and the

engineer agent may not be able to solve it, at least immediately and without spending time learning. If the agent were not capable of solving the problem within the constraints imposed by the project, Rule #5 would be followed by Rule #6 and possibly Rule #7.

Rule #2 is less generic than Rule #1, but it triggers an action that depends on the capacity of the agent tested by Rule #5 and possibly followed by Rule #6 and Rule #7.

Rule #4 is very specific and should be actioned by a very specific agent: electrical engineer experienced in ultra-sound interfaces. Specific rules should be included in the performance system of agents with the capacity to execute the action determined by the rule.

Table 7 – Sub-Set of Rules in an Engineer Agent Performance System

Rule #		Condition		Action
1	IF	< presented with a problem>	THEN	<solve it>
2	IF	<presented with an engineering problem>	THEN	<solve it>
3	IF	<presented with an electrical engineering problem>	THEN	<solve it>
4	IF	<presented with an electrical interface problem with ultra-sound sensor >	THEN	<solve it>
5	IF	<I don't know how to solve the problem>	THEN	<get a consultant>
6	IF	<consultant is not available>	THEN	<learn how to solve the problem>
7	IF	<I don't have time to spend on learning>	THEN	<ask for help>

Agents prefer specific rules because these rules often require less investigation and trigger actions that are in the agent's area of expertise and within their comfort zone. Generic rules are more difficult to handle as they require further investigation, testing, possible learning and interaction with other agents. Generic rules, however, are more suitable for situations that have not been experienced before and can be used to reduce the number of specific rules that are applicable to similar situations.

Credit Assignment

Credit assignment is the process envisaged by Holland (1995) that allows the agent to evaluate the effectiveness of a rule. Every time a rule is applied the agent assesses its effectiveness and assigns a credit to the rule. If the action triggered by the rule leads to a successful result the credit is increased, otherwise the credit is decreased. This way,

before choosing a rule the agent checks its credit and chooses the rule with the highest credit for that situation. The process of credit assignment can promote some rules as the most effective and totally ban rules that have been shown ineffective. The process will be effective as long as the agent retains the information about previous experiences. Situations that do not happen very often may be forgotten and the agent may repeat unsuccessful rules because the information of their credit was not retained.

Continuing the example of possible rules in an engineer agent performance system, shown in Table 8, the credit assignment process is illustrated. The engineer agent has realised that they do not possess the knowledge and expertise to solve the problem (positive answer to Rule #5) and that a consultant is not available (positive answer to Rule #6). The possible actions from Rule #6, and Rules #7 to #9 inclusive, are respectively: ‘learn to solve the problem’; ‘ask for help’; and if these are inappropriate, invoke Rule #10: ‘try your best without spending time learning’. The actions from these rules could lead to a successful or unsuccessful outcome.

Table 8 – Another Sub-Set of Rules in an Engineer Agent Performance System

Rule #		Condition		Action
5	IF	<I don't know how to solve the problem>	THEN	<get a consultant>
6	IF	<consultant is not available>	THEN	<learn how to solve the problem>
7	IF	<I don't have time to spend on learning>	THEN	<ask for help>
8	IF	<Agent A is available>	THEN	<ask for help>
9	IF	<Agent B is available>	THEN	<ask for help>
10	IF	<There is no one to help>	THEN	<try your best without spending time learning>

Suppose that the project is late and management direct the team not to spend time in ‘non-productive tasks such as learning’. The engineer agent will then attempt to get some help, first using Rule #8. Agent A however, is already busy and not willing to help. The ineffectiveness causes the engineer agent to decrease the credit of Rule #8 and try Rule #9. Agent B is very constructive and willing to help. The engineer agent gets what they want and as Rule #9 is shown to be effective, it gets its credit increased. As a consequence, next time the engineer agent is looking for help it will try Rule #9 before Rule #8. If in future opportunities to invoke Rule #8, asking help from Agent A fails

again, this rule will have its credit further reduced and may even be removed from the performance system.

Suppose that both Rules #8 and #9 are shown to be ineffective (every agent in the system is busy and unwilling to help) and the engineer agent has to execute the action from Rule #10: ‘try your best to solve the problem without spending time learning’. If the problem is solved in accordance with the action from Rule #10 (the engineer agent was lucky and managed to solve the problem within the time constraints), the credit for Rule #10 increases and the engineer agent will be confident of using Rule #10 again. Rule #10 may even develop a higher credit than Rules #6 and #7, and the engineer agent may prefer in the future to try their luck attempting to solve the problem without learning or asking for help. Alternatively, if the action from Rule #10 fails and the engineer agent was unable to solve the problem within the time constraints, the credit of Rule #10 decreases and the rule may be avoided next time. The engineer agent would have to balance the consequences of ‘trying its best without learning’, ‘asking for help in a likely unfriendly environment’ or ‘spending some time learning how to solve the problem’. Each of these actions may have good and bad consequences. Trying to solve the problem without learning may result in successful or unsuccessful results, and the same could happen if the agent spends time learning, breaching management instructions.

Intelligent adaptive agents always have decision-making power within their realm of actions. The course of action chosen by the agent leads to consequences that could praise or punish the agent and determines the agent’s path to adaptation.

Rule Discovering

To complete the model of an adaptive agent Holland discusses the need for a process of rule discovering that is a process to create new rules when the existing ones do not fit the purpose of the current situation. The first possible technique is the ‘random trial and error’: given a situation, try some action (whatever the action is) and check for the effectiveness of the rule and associated action. The random trial and error is a risky and most likely inefficient technique, but it could be the only option for an agent starting with a performance system with an empty set of rules.

Another and probably more efficient technique is to reuse, adapt and combine existing rules: the agent assesses the situation and searches the set of rules for existing rules that

could be helpful to address the situation; the chosen rules could be modified and combined to create new rules. This technique of rule discovery benefits from existing generic rules and building blocks discussed previously in this and previous sections.

Social systems of engineering projects are designed taking advantage of this technique. Engineering projects have engineer agents, i.e. agents that are trained in the engineering disciplines and ideally have experience in the fields of engineering that facilitate the solution of problems presented by the project. Engineer agents bring a ready to use performance system comprising useful and likely efficient, although generic rules. This way, engineer agents will not have to adopt random rules to solve the problems presented to them, which could not be the case if instead of engineer agents the project had hired ‘lawyer agents’ to perform engineering tasks.

The initial performance system brought by engineer agents is often increased by other rules imposed by the engineering project. A typical example is the inclusion of rules on how to apply processes and procedures that are specific to the project and to the organisation. Other rules could be included by additional technical knowledge specific to the project acquired by the engineer agent through specific training and learning during the initial phases of the project.

In addition to this initial augmentation of the performance system, the engineer agent will continue adapting and enhancing its set of rules throughout the life of the project, using building blocks and similarities to combine and modify existing rules and in this way creating new rules. The initial process system, even after being enhanced at the project start-up, will still contain a considerable number of generic rules that will need to be tailored to become useful and effective for the project.

The same process of augmenting and adapting the performance system happens to other kinds of agents in the system: managers, leaders, technicians, writers and other support agents.

3.6. Emergent Properties

Organisations are social complex adaptive systems that depend on emergent properties to achieve their goals. Software-intensive acquisitions, as social complex adaptive systems, as discussed in Section 3.5, present these characteristics.

Emergent properties that are desirable to the software-intensive acquisition system are those that add value to the intended objectives: deliver a solution that satisfies the need within the established constraints of cost, schedule and quality. It is also reasonable to assume that the design should attempt to avoid emergent properties that would move the organisation away from the desired objectives. Undesirable emergent properties often cause software acquisitions to miss parameters of cost, schedule and quality and may even cause the acquisition to be cancelled; a project failure.

Value may be in the form of explicit and implicit goals. For the customer explicit values are associated with the solution that satisfies the need and meets the established budget and time constraints. There are however also implicit values: the benefit that the solution will bring; political implications if the project succeeds; and personal gain for stakeholders in the customer organisation. For the provider, explicit values are the same as for the customer that is developing a solution that meets the customer's expectations of cost, schedule and quality, which in turn should also meet the provider's organisation business needs of profitability and sustainability. The provider's organisation also has implicit values; for example, the maximisation of profitability and increase in the company's value; the maintenance of sustainable business and increase market share; the generation of personal gain for stakeholders in the provider's organisation.

The three desirable emergent properties of organisations (quality, knowledge and core competency) introduced in Section 2.4.6 will be discussed in this section within the context of software-intensive acquisitions.

3.6.1. Quality as an Emergent Property

Often quality is associated with the product delivered to the customer. Quality has multiple dimensions related to implicit and explicit values and to both the customer and provider organisations. In broad terms quality can be associated with the effectiveness and efficiency of the organisation. Effectiveness determines how the organisation, or the system, achieves its intended purpose, meaning that high effectiveness is an indication that the organisation achieves the intended objectives. Efficiency is an indication of how much waste is produced in the process. High efficiency results in lower costs to the customer and higher profit to the provider. Efficiency, however, is meaningful only if the system is effective. Quality is therefore a desirable property that should emerge in software-intensive acquisitions systems.

The Japanese Lean Production Experience

A classic example of quality as an emergent property of the system is the lean production of Japanese companies (Rechtin 2000). In the 1950's Japanese companies embraced a number of quality improvement approaches proposed by Deming. Although the application of quality improvement techniques in isolation was not able to produce the expected and desirable results, high quality emerged when several techniques were applied together and consistently across multiple organisations. Experienced-based principles like the recursive approach of the 'five whys' (why this happens; then why that happens; then why this happens; ... five times); and the notion that 'everyone is a customer, everyone is a supplier', created a culture that everyone should produce products of acceptable quality for their customers and expect nothing less than acceptable quality products from their suppliers, as well as developing continuous improvement, seeking the root cause of defects and actively resolving them.

Rechtin explains that the moment was also right. The post-war period created the motivation to rebuild the country and to show to the world what Japan was capable of. These drivers were instruments of vision, method and action. The system by itself without governance most likely would not be able to find its purpose and the way to achieve it. The Japanese government provided the vision and policies that enabled the Japanese industry to find the way that was good for everyone. There were no losers, only winners. In Japan, as a more collective society, motivation exceeded individual interests and benefitted all.

The good result of the lean production experience in Japan was possible because the approach was applied with determination and consistency across the system whose boundaries extended not to just one company and not even the whole industry, but the entire country of Japan (Deming 2000 p. 55). Deming explains (2000 p. 58) that the same techniques can also be applied in service organisations.

The Development of On-Board Software for the Boeing 777

The success story of the development of the on-board software for the Boeing 777 shows another case of quality as an emergent property of the system, and directly related to the topic of this research as a software-intensive acquisition of a commercial airline aircraft (DAF 2000 Appendix Q).

To give an idea of the order of magnitude of the Boeing 777 on-board software, about 2.5 million source lines of code was newly written, which is six times the amount of software developed for the 747-400, then the most recent aircraft developed by Boeing. Over four million lines of code of commercial off-the-shelf software (COTS) were integrated into the software to implement optional features. The software was partitioned into more than one hundred major components, corresponding to physical boxes distributed in the airplane's control system. These components are capable of processing of the order of 3,000 analogue signals and 40,000 digital signals.

Various suppliers external to Boeing Commercial Airplane Group (BCAG) developed the software. The Airplane Information Management System (AIMS) was developed and supplied by Honeywell, which provides several functions including primary pilot displays, flight management, thrust management, data acquisition and airplane condition monitoring and central maintenance.

Driven by the motivation to succeed in the 777 Program, Boeing created the vision for success starting from what seems to be the ‘severe software-development impediments’, shown below:

*Severe software-development impediments, as seen by Boeing (BCAG)
(DAF 2000 p. Q-4):*

- *Poor communication among groups working on the same project;*
- *Lack of understanding in system requirements on part of customers and designers;*
- *Difficulty in monitoring progress in a software project, since program construction is not always a simple progression in which each act of assembly represents a distinct forward step;*
- *Lack of inventories of reusable software components;*
- *Rapid growth in size of software systems.*

The plan of action was based on a communication which established ‘Working Together’ as the slogan of the 777 Programs. The plan of action also included several

well-known software and systems engineering approaches: partition the system into well-defined pieces with well-defined and simple interfaces among them; for each system, a System Requirements and Objectives (SR&O) document, setting forth high-level requirements; each SR&O was the basis for the Specification Control Document (SCD) defining the requirements for each system; a single Interface Control Document (ICD) database; catch defects, particularly defects in requirements, before moving to the next phase, and so on. The 777 Program also applied extensively the use of tools and simulation to reduce risks and errors normally caused by ordinary mechanical tasks.

Customer involvement and real situation testing played an important part in the plan of action to success. At the time of delivery, the airplane had completed 1,751 test flights in a total of 3,379 hours, and 90 flights had been flown by United Airlines to simulate day-to-day flight maintenance and operations procedures.

The approach envisaged and implemented by Boeing, although perceived as sensible, could not guarantee success. Boeing was aware of the complexity of the task and of the system assigned to make it happen. The ‘working-together approach’ provided the flexibility required to cope with the unknown. Most participants in the 777 program believed that the working-together approach was an absolute prerequisite for success. Boeing, its customers and its suppliers worked solutions to the software development problems and impediments. Tools and techniques from engineering best practices would not have produced the same results if applied in isolation and without the collaborative environment created by the 777 Program by the working-together approach.

The several tools and techniques applied in the 777 program ‘should be seen as viable means to accomplish established software objectives, rather than magic bullets’ (DAF 2000 p. Q-4), in a clear allusion to the assertion made by Fredrick Brooks in ‘The Mythical Man-Month’ (Brooks 1995) that there is ‘no silver bullet’ to succeed in software development.

The first Boeing 777-200 was delivered on schedule and accepted by United Airlines in May 1995, four and one-quarter years (51 months) after Boeing had committed to the 777 Program. Lew Kosich, United Airlines fleet captain stated:

I've never seen a program go like this. Working together was absolutely awesome. When the airplane was finally inaugurated into service June 7, it

was almost a non-event for us. We were inaugurating a mature airplane.

(Kosich in DAF 2000 p. Q-5)

The success of the Boeing 777 Program is the result of the high quality that emerged from the system comprising a network of individuals and organisations working as customers and suppliers, motivated and managed by Boeing which was also a customer as well as a supplier.

The Development of Linux

Linux is a Unix-like operating system developed as a collaborative task under the ‘open source’²⁷ movement, which means its source code is freely distributed and available to the general public. The revolutionary method used to develop Linux produced a robust product that has proven to be of good quality, free of cost to the user, and became widely accepted to the point that it is threatening the interests of giants like Microsoft (Moody 1998). Linux OS and its development method started from the initiative of one individual, Linus Torvalds²⁸, driven by the personal motivation to develop a Unix-like OS.

Linux is a complex technical system, developed by a not less complex social system: the Linux community. The development of Linux is an example of software development as a complex adaptive system, where software developers execute their tasks, interact and adapt, without a central control. The success of the development of Linux can be attributed to the product quality that emerged from the system that constitutes the Linux community.

The open source software development approach worked well for the development of Linux, and created a complex adaptive system that is effective and robust. The success can be seen by the high quality of the Linux OS, its wide acceptance and the satisfaction of the members of the Linux community that continue offering their work for free. The social system that develops Linux is also robust. It adapts to changes that may occur in the pool of contributors and resists threats that come from large organisations in the

²⁷ ‘Open source’ does not refer to the Open Source Initiative (OSI). Linux is written and distributed under the GNU General Public License, under the banner of the Free Software Foundation (FSF).

²⁸ Linus B. Torvalds, born in 1969, Helsinki, Finland, software engineer, started in the early 1990’s the development of a Unix-like kernel that became Linux OS (source Wikipedia, http://en.wikipedia.org/wiki/Linus_Torvalds).

business world. For over 15 years the Linux community has survived and continues achieving its goal: to produce software of high quality, flexible and at low cost, at least for the users.

Quality as Emergent Property Explained

Quality as an emergent property of the system that developed the Boeing 777 and Linux can be explained by insights from complexity theory, and may help to find strategies with better chances of success for software-intensive projects. Three key processes provide the basis for adaptation: variety (or diversity), interaction and selection (Axelrod & Cohen 2000).

Boeing 777

The three key processes for adaptation were present in the development of the Boeing 777. The identification of the ‘severe software-development impediments’ defined the selection process that created the basic rules to avoid and resolve the impediments. Variety was present in the form of the knowledge, expertise and experience of customers and providers. The interaction process was also taken into consideration through the ‘working-together approach’ which not only improved communication, but also motivated all to resolve the impediments that could prevent the success of the project.

It is also interesting to observe that the project achieved all the three dimensions of cost, schedule and quality. The explanation may arise from the fact that management and the leaders of the project were aware of the main impediments and the intrinsic limitations of the system, as if they had a view from outside the system, and created the conditions for the system to do what was necessary to achieve the intended and explicit goals. The system was steered to the desired course without being placed under a forced control.

Linux

Variety is present in the Linux community in the abundance of contributors, which possess two important attributes: motivation and knowledge. People that join the Linux community do it because they want to and feel confident that they can contribute with their knowledge, experience and skills. Their motivation may be driven by different factors, whether making better software they can use, to expand their technical knowledge, or because they want to oppose the hegemony of large business-driven

corporations. Their goal, however, is the same and determined by the open source and free software initiatives.

The participants in the Linux community interact through a strict protocol and follow the etiquette of the open source community. Error reporting, suggestions and fixes are submitted via a well-defined process. Selection occurs by the intervention of a select team that decides what will be included in the next release of Linux. Linux's error reporting and selection processes are a form of Change and Configuration Management Control suggested by good software engineering processes. However, in the case of Linux, error reporting and section processes are likely emergent properties of the Linux community instead of imposed by planning in advance.

Linux is developed by volunteers, and there is no concept of individual gain, except for the satisfaction of doing what they want and believe, and perhaps to increase status in the community. The Linux community works as a team towards a goal that benefits the community and not just individuals, which is the essence of complex adaptive systems.

The Real Costs of Linux

The real costs of Linux and the effort spent in the development are unclear. The cost of Linux Kernel 2.6 is reported to be between US\$50 million and US\$176 million and the cost of the whole Linux distribution, including Kernel, API Library and Core OS Utilities, is estimated in the order of US\$1 billion (Unknown Reader 1998; Wheeler 2006).

The lower estimated cost for Linux Kernel 2.6 seems to be based on the average industry standard cost to produce the same number of SLOCs while the higher estimation assumes that if Linux were developed from scratch it would take at least the same effort as a similar commercial project estimated using the default COCOMO model (Wheeler 2006).

Both estimations and their assumptions are unlikely to be valid for Linux. A horde of highly motivated programmers working part-time and for free is unlikely to have the same productivity of an average programmer in the industry. Also, the estimations seem not to take into account that the development of Linux is highly redundant and only a portion (the good one) of the code developed is selected to be included in the official release of Linux. The higher estimation, however, is an indication that the performance

in the development of Linux could be at least 3.5 times less than the average industry standard. Taking into account the software code that is developed but not included in the product, the real cost of Linux development could be much higher, lowering even more the efficiency of the software development process of Linux, although there is no data to confirm this hypothesis.

Efficiency versus Robustness of Linux Development

The open source software development approach worked well for the development of Linux, and created a complex adaptive system that is very successful and robust. The success can be seen by the quality of the Linux OS, its wide acceptance and the satisfaction of the members of the Linux community that continue offering their work for free. The social system is also robust. It adapts to changes that may occur in the pool of contributors and resists threats that come from large organisations in the business world. For over 15 years the Linux community has survived and continues achieving its goal: to produce software of high quality, flexible and at low cost (at least for the users).

As the complexity of technical systems increases, the social system that engineers the solution needs to be robust to cope with the unknown, which in turn reduces the nominal efficiency of the system to achieve the intended effectiveness. The high robustness and possible low efficiency of the development of Linux confirm that systems that are robust, trade robustness for efficiency (Hitchins 1992 p. 89).

The original refereed paper published by this research arguing that better strategies are needed for balancing efficiency and robustness in software-intensive acquisitions, is included in the Appendix F.

Discussion

The three cases of quality as an emergent property of the system show the same fundamental characteristic: motivation originates the vision that guided the creation of a plan of actions. The vision and plan of actions, however, had no guarantee of success. The awareness and recognition of this fact was apparent in the Boeing 777 Program, although it was not apparent in the lean production experience in Japan and most likely completely oblivious in the case of the development of Linux.

Although it can only be speculated, it seems that the success experienced in both cases of lean production and Linux seem to have been unexpected or perhaps well beyond

expectations. The Boeing 777 Program, however, shows signs that it was conscientiously and carefully planned and thus the results were more as expected, although it may have also been beyond the expectations.

It is interesting to observe that there is a trade-off between effectiveness and efficiency. The Linux community was effective in developing Linux at the expense of low efficiency created by a highly redundant system. The Boeing 777 Program was also effective in developing the on-board software for the aircraft, although the effort applied to tasks that did not contribute directly to the final product, such as the development of supporting tools and simulation, can be seen as a reduction in efficiency. One could argue that when the system is better prepared to perform the task, less effort is spent on supporting tasks. However, the more complex the task, the less are the chances the system is fully prepared to accomplish it; and in this case support tasks that mitigate the risks should be justifiable as a reduction of efficiency.

The Boeing 777 Program experience shows that it is possible to design software-intensive acquisitions in ways that high quality is likely to emerge as a property of the system. The key aspect for the success of the Boeing 777 Program is that the vision, the plan of actions and its management originated at the top level of the system, the customer. Boeing, as the customer, managed to propagate its vision, motivation and plan of actions to the chain of suppliers that together with Boeing and the user, United Airlines, developed the on-board software for the Boeing 777, the object of the software-intensive acquisition.

It is reasonable to assert that every organisation that initiates and is responsible for the success of software-intensive acquisitions will claim that they also have the same objectives and their programs are structured and planned to succeed. Many of these same organisations will also admit that their efforts led to less than the desirable results. These organisations could compare their programs against the 777 Program and identify the differences that contributed to success and failure.

3.6.2. Knowledge as an Emergent Property

Knowledge is a necessary condition for software-intensive acquisitions to succeed, as discussed in Section 3.4. Knowledge and knowing how to use it is a new form of wealth and has become one of the central purposes of the organisation (Rechtin 2000). Knowledge can be acquired, shared and created. In fact knowledge can create more

knowledge. Given the right conditions, knowledge can emerge in the organisation, hence the call by Senge for the Learning Organisation (Senge 2006).

A Case of Emergent Design Solution

CSC²⁹ Australia developed the Combat System Simulator and Trainer for the ANZAC class frigates. The simulator comprises a simulation of the ownship³⁰ and its equipment (navigation sensors, tactical sensors and weapons) and the tactical environment (weather conditions, vehicles in the tactical environment and their emissions). Radars are an important component in the simulation of the ownship's navigation and tactical sensors. The simulation of radars includes the generation of the video signal that is displayed in real radar consoles. The design of the radar simulation was developed in the early 1990's and made use of existing COTS video cards that allowed the video parameters to be programmed into the card and in this way to produce the video signal required for the radar consoles.

Over the years it became clear that the radar video graphics (RVG) hardware presented a maintenance risk because the manufacturer no longer supported the original video card. A need was identified to find another solution for the RVG hardware. As the original hardware was still operational, and some spare parts were available, the solution for the problem was not urgent. However, an initial analysis was performed to identify possible solutions. Informally two solutions were identified: (a) a dumb digital to analogue (D/A) interface controlled by the radar simulation computer; (b) a smart video interface based on Field Programmable Gate Array (FPGA), also controlled by the radar simulation computer. The option using a D/A interface was identified as impractical because the D/A available commercially at that time could not support the data transfer rate required by the RVG specification, and the FPGA solution imposed a high risk due to the lack of experience in that area. The problem and its possible solutions remained dormant for years until the day there was a real need to replace the navigation radar.

²⁹ CSC is a global business solutions and technology services provider with over 93,000 employees in 90 countries (source Wikipedia, http://en.wikipedia.org/wiki/Computer_Sciences_Corporation). CSC has been present in Australia since 1970 and has been involved in several Australian Defence projects including the Collins Class Submarine, the Combat System Simulator and Trainer for the ANZAC class frigates and the Super Seasprite Helicopter.

³⁰ Ownship is the main vessel in the simulation environment in which the crew is being trained.

When CSC was awarded the contract to upgrade the ANZAC Combat System Simulator to replace the navigation radar the project had to find an urgent solution to replace the RVG hardware. At the time the author was the Systems Engineering lead for the project and he was aware of two possible solutions. However, he knew that the solutions had been suggested several years earlier and could be out of date, and it would be good to investigate whether other options existed. The situation offered an opportunity for exploration and experimentation with teamwork and cooperation as well as competition.

A task was commenced to explore other solutions. Two engineers, one junior and the other senior, both with knowledge of hardware and software, were asked to analyse the problem and propose solutions. Both engineers would work separately and independently. The two engineers were asked not to share their findings. The intention was to accentuate variety in solving the problem. The two engineers were also instructed not to be biased towards the solutions that had been identified before. After a given time these two engineers would present their proposed solutions for discussion to an audience knowledgeable of hardware, software as well as radar simulation. The two engineers were instructed to compete with their solutions until they were presented and discussed. When the winning solution was selected they should then cooperate to implement it. This could have been a dangerous approach if the team had been newly formed. However, the two engineers had worked together before and the approach was seen as a low and manageable risk.

During a couple of weeks the two engineers worked hard in the attempt to produce the best options and a climate was created that on the ‘big day’ they would try to convince the audience that their personal solution was the best and should be adopted.

On the day of the presentation both engineers were ready and very eager to show the result of their efforts. As they presented their options the result was somehow disappointing, as both solutions showed the same options identified earlier. Both presented the FPGA solution and one of them offered a solution based on the D/A interface and they reported the same risks and limitations identified before, yet the FPGA-based solution was considered too risky due to the team’s lack of experience in developing a not so simple FPGA software. As expected, the two engineers proposed other solutions that due to cost and other limitations were discarded. The D/A solution, although simpler was impractical, while the FPGA option presented higher potential but

imposed risks. At the end of the meeting, it was agreed that the FPGA solution although risky was the only available option. Apparently the resort to variety was not enough to produce other attractive solutions.

Surprisingly, a third approach arose soon after the meeting during a discussion at the coffee machine. It was suggested that a more likely solution would be in combining both the FPGA and the D/A solutions. FPGA hardware available commercially includes D/A converters that support the output rate required, so a solution was to use FPGA to implement the D/A interface. This way, the complexity of the FPGA software would be reduced and moved to the host computer, which is within the team experience. The D/A-FPGA solution was born and would be the bridge between the existing and the new technology. As the team acquired experience with FPGA, the sophistication of the FPGA software could increase to reduce the processing load on the host computer.

When asked about how the new solution was created, the engineer that had proposed the FPGA and not the D/A solution said that he had thought about it and in fact one of his proposed options was doing the same thing. It was argued that he had not been clear at his presentation nor was his written report specific about this option. He agreed but insisted that he had thought about it. Another senior engineer that had investigated the problem years earlier said that this solution was always there. Again it was argued that the D/A-FPGA solution had not been specifically conveyed. The engineer then suggested that the solution was ‘hovering around’ and it was just a matter of naming it.

While the idea is in one’s head and has not been explicitly conveyed it may be ‘hovering around’ but it does not exist. In the moment it was verbally communicated the RVG D/A-FPGA solution was set to become a reality. The engineers stated that the new solution was too obvious to have been missed and no one could assume having overlooked it. Most likely the engineers had really thought about it but only the discussion at the team level caused the obvious solution to become reality.

The RVG D/A-FPGA solution was the result of knowledge created as an emergent property of the team. As such, the emergent property cannot be predicted and there is no guarantee of success. The team possessed an implicit variety that was motivated to be applied through two independent proposals; there was an interaction process where the two proposals were presented and discussed; and a selection mechanism that caused the two initially competing engineers to cooperate and produce a combined and new

solution. The result of the experiment was successful and demonstrated on a small scale how social design can be applied to create knowledge as an emergent property. The result, as expected, cannot be controlled but can be ‘steered’ to the desired and explicit outcome.

3.6.3. Competency as an Emergent Property

Rechtin (2000) defines competency as the emergent property that makes the organisation more than buildings, processes and the statement of the organisation’s mission; the core competencies of the organisation are determined by the core competencies of its people.

Elaborating on Rechtin’s concept, the core competencies of the organisation emerge when the people in the organisation believe and live the organisation’s mission and vision, reflecting an alignment between implicit and explicit goals of the organisation and its members. The emergence of the organisation’s core competencies reflects a mutual trust between the organisation and its members. As an emergent property, competency is not imposed; and it does not come free either. It may take a long time and an innovative environment to emerge, and requires commitment, effort and resources to be maintained. Some organisations may never achieve it.

Employees in organisations that demonstrate strong competency would be willing to go beyond their duties to achieve a difficult goal; they expect to be realistically challenged in pushing their boundaries. Employees also expect some form of fair recognition for their extra effort, whether by implicit or explicit rewards, or by financial compensation; and they know when they are being abused. As trust can be lost, so can the values that competency brings to the organisation. Competency can be damaged when the organisation exploits its employees without compensation or recognition; sets unrealistic targets; or demands extreme personal sacrifice to achieve impossible or quasi-impossible targets.

As the market and other conditions in the environment where the organisation operates change, the organisation has to adapt its vision, strategies and competency. Competency is therefore an emergent property that reflects how well the organisation moves in its fitness landscape in searching for better positions.

3.6.4. The Ultimate Emergent Property

Competency, knowledge and quality are desirable emergent properties that alone may not mean much. What would be the use of competency if there is no challenge to achieve? What would be the use of knowledge if it were not applied to create quality products? Without knowledge and a motivation to apply the organisation's competency there would be no quality. Quality therefore emerges as a result of competency and knowledge. The three properties are interconnected as a system. Competency fuels the motivation to learn and make things happen; learning produces knowledge that when applied creates quality products; when the organisation is capable of creating quality products its members will believe in the organisation, and this is the central aspect of competency.

The ultimate emergent property is what causes the organisation to achieve its goals and fulfill its mission and is achieved as an emergent result of three other emergent properties, competency, knowledge and quality, working as a system. This reasoning conforms to Deming's reporting of what ignited Japan in the 1950's resulting in the revolution of Japanese production (Deming 2000 pp. 57–59).

3.6.5. Undesirable Emergent Properties

Not all emergent properties are good and some can destroy the enterprise and the organisation. Just as quality products emerge as the result of competency, and knowledge, deficient products emerge in the absence of these properties. Defects and products that don't satisfy the need take time to develop and likely cost more than would the just right solution. When inadequate solutions emerge, other undesirable properties are also likely to emerge: tension between customer and contractor; aggressive management; passive staff not willing to learn and cooperate, among many others. These negative properties would cause the system to meet its intrinsic aim, i.e. waste time and resources without achieving the intended aim, and ultimately collapse, unless the system is changed through leadership and management intervention.

3.7. Systemic Structure

Systemic structure, introduced in the review of the literature in Section 2.3.4, defines the interrelationship of key variables that determines the behaviour of the system over

time. Systemic structure defines the system itself. Social and technical aspects determine the systemic structure of software-intensive acquisitions.

The systemic structure of software-intensive acquisitions is determined by two co-existing systems: the technical system and the social system. The technical system does not present behaviour and it is not capable of changing itself. The technical system is created and modified by the social system. The technical system would not exist without the social system. The systemic structure of the technical system however, influences the behaviour of the social system. The social system would have no purpose without the technical system and without which the social system would cease to exist.

3.7.1. Technical System

The need and the software-intensive solution define the systemic structure of the technical system, and exists in the form of artefacts applied to describe the need and to develop and implement the solution. Artefacts are documents, software and other forms of technology. Documents are created and used by the social system. Technology is used in the development process. Software is part of the final product delivered to the customer together with other artefacts that together comprise the solution.

Artefacts in the technical system are interdependent. Specification influences design, and design influences the implementation. Interaction between artefacts is a relationship of dependency. The nature of the need and the required solution determine the complexity and the knowledge that is needed to understand the need and to realise the solution. The systemic structure of the technical system is determined by artefacts and their interdependencies; by the tasks that are needed to engineer and develop artefacts; by the knowledge to execute engineering and development tasks; and by the complexity to understand the technical system and its components, and the uncertainty created by what is not known.

The required knowledge and complexity determine the type of professionals, their skills and experience that are required to engineer the solution and to manage the software-intensive acquisition enterprise. The tasks to engineer the system and to manage the enterprise will drive individual and group behaviour in the social system. The behaviour of the social system is also influenced by the performance of the technical system in positive and negative ways. If the solution under development is meeting expectations of quality, cost and schedule, people in the social system will get a feeling of

achievement which influences constructive and learning behaviours that makes the social system perform even better. However, when the development of the technical system does not meet expectations, whether by not meeting plans and budgets or technical performance, the social system could respond with constructive behaviours that may improve the performance of the system or with aggressive-passive behaviours that most likely will worsen the situation.

3.7.2. Social System

The social system influences and is influenced by the technical system. The systemic structure of the social system is partially determined by people, organisations in their roles and responsibilities; by artefacts that define plans and conditions of contract; tasks that connect people and artefacts; and by the relationships that connect people and organisations. This is the visible and explicit portion of systemic structure.

The software-intensive acquisition, as a social system, has many other attributes that influence the behaviour of the system and therefore part of the systemic structure. These attributes are driven by informal and implicit human behaviour and can never be completely understood. The best that can be done to attempt to understand these aspects of the system is to apply knowledge of psychology and social behaviour, some of which was introduced in the review of the literature in Section 2.4, and make assumptions about what may influence this side of the systemic structure.

The systemic structure of the social system is also likely to be influenced by motivation and goals of individuals and organisations; by individual knowledge, skills and experiences; by individual personalities and behavioural styles; by the culture of organisations; by individual and collective tacit knowledge; by individual preferences and moods; and by informal relationships like friendship and acquaintances. The list can become very long and will never be complete for there are unknown factors yet to be discovered and understood. The relevance however, is that there are too many factors, unplanned and unknown, that influence the outcomes of tasks and the collective outcome of the acquisition, and the way they interact in response to the state of the acquisition is very complex.

3.7.3. Underlying Processes

The systemic structure of complex systems is also complex. Complexity produces effects that cannot be predicted, much less controlled. Complex social systems however, can be steered towards intended goals, as long as that is within the limits of what the system can do.

The ‘Framework for Harnessing Complexity’ introduced in Section 2.3.6 helps to reveal and understand the underlying processes that create systemic structures that impel the system towards or away from intended goals. The ‘Framework for Harnessing Complexity’ suggests three key interlocked processes: *variation*, *interaction* and *selection*.

Variation

Variation in software-intensive acquisitions is present in both technical and social systems. In the technical system variation is in the diversity of artefacts, knowledge and technology that is present in the need and in the software-intensive solution.

Variation in the social system is present in people and organisations and in their knowledge, skills, and experience. Variation is also present and in the set of strategies that people and organisations apply to realise the technical systems. Strategies are in the processes, life cycle models, contract models, reward and penalty models and in the time and effort applied to realise the technical system.

The variation available in the social system has to be sufficient to match the variation in the technical system, also in accordance with the ‘Law of Requisite Variety’ (see Sections 2.3.1 and 3.2.1), i.e. knowledge, skills, experience and strategies available in the social system must be at least sufficient to specify the need and develop a solution for that need.

Interaction

Interaction connects people, organisations and artefacts in the system. There three basic types of interactions: *interaction*, *task* and *dependency*.

Interaction

The term *interaction* will be used to express the interaction between people and interaction between organisations. Interaction can be formal or informal, explicit or implicit.

Formal interactions occur through channels created by the structure of the organisation defined in the organisational chart, and establish relationships defined by roles and responsibilities, e.g. customer and contractor; programme manager and managers; manager and team leaders; team leader and team members; experts and team members, to name a few. Informal interactions are established by friendships and acquaintances and do not convey formalities of roles and responsibilities. Informal interactions are good channels to transfer information, knowledge and motivation. Formal and informal interactions are explicit and occur through formal or informal, written or verbal communications. Explicit interactions are vehicles to transfer information and knowledge and can influence change in behaviour and motivation.

Implicit interactions occur without written or verbal communication, through expressions and feelings. A person can implicitly interact with another person and with the environment, by sensing and making and assessment of the situation. Implicit and explicit interactions can also convey information and knowledge but more often implicit interactions influence change in behaviour and motivation.

Task

Task is the interaction between a person and an artefact. Through a task a person can create, modify, review or obtain information from an artefact. Tasks assigned to organisations, groups and teams need to be partitioned to the level that it becomes an interaction between a person and an artefact. Tasks are the mechanisms through which the social system interacts with the technical system. Tasks have an associated level of effort, duration and knowledge that is needed to execute the task.

Dependency

Dependency is a passive interaction between artefacts and is determined by the systemic structure of the technical system. The code that implements a particular function has a dependency on the artefacts that define the design and software requirements, and these have a dependency on artefacts that defined the system architecture and system

requirements. Changes in higher-level artefacts will influence changes in lower-level artefacts if they are connected through dependencies. Tasks need to take into consideration the dependencies between artefacts, and the people that execute the tasks should be aware of these dependencies.

Selection

Selection is the process that defines what strategies work best and are defined by measures of success. The contract is the formal and explicit vehicle that defines the reward and penalty models and measures of success. However there might be implicit and hidden measures of success and rewards that conflict with the contract. In large software-intensive acquisitions political and commercial interests play important roles in the reward/penalty model. The best strategy will be the one that produces the best balance of explicit and implicit reward.

Implicit measures of success and rewards that conflict with the intended aim of the software-intensive acquisition is one of the most prevalent causes of a bad systemic structure. If the conflict is not resolved the acquisition is destined to failure.

3.7.4. Good and Bad Structures

Good structures are those that induce the system to develop emergent properties that will lead the system to achieve the intended aim. Conversely, bad structures will move the system away from the intended goals. The systemic structure defines the system and what it does, and if that does not match the intended aim, the systemic structure, i.e. the system, has to change.

Good structures align the capacity of the social system with what is needed to realise the technical system, or include mechanisms so that the alignment emerges as a property of the system. If the systemic structure does not allow this to happen, the acquisition is destined to failure. The first challenge is to obtain sufficient information about both the technical and the social systems, which is not always possible, in which case the design of the systemic structure will carry uncertainties.

It can be hypothesised that systemic structures that facilitate constructive and learning behaviours impact positively to the performance and effectiveness of the system. Conversely systemic structures that promote aggressive-passive behaviours should contribute negatively to the performance and effectiveness of the system.

The Super Seasprite Helicopter Programme

The Super Seasprite Helicopter programme is a real and recent case of complete failure of a software-intensive acquisition in Australia (see Section 1.4.2). The failure was the result of many factors, some created by the influence of political considerations in the ADF and Department of Defence decision-making process, including: starting with the specification of the need for a lightweight aircraft that drove the development of a new and complex two-man crew avionics system; lack of experience of the prime contractor in handling a major sub-contractor in this kind of project; a fixed-priced contact that did not include provisions for liquidated damages in case the prime or sub-contractors failed to perform adequately; lack of action by the Commonwealth to address the warnings and clear difficulties in the course of the project (Ferguson & Blenkin 2008).

The failure of the Super Seasprite Helicopter programme is a case of negative properties that emerged as a result of overestimating the need requirements and underestimating what would be needed to develop the solution with the experience and expertise available. The project could have been classified at the highest level of Order of Ignorance, 4OI (see Section 2.4.7, Table 5).

3.8. Hypotheses about the System

After the appreciation of the software-intensive acquisition system two hypotheses can be formulated in response to the question raised by the first research sub-problem, stated below.

What can be learned about the software-intensive acquisition system that would help to understand the system's ability in achieving its intended goal?

3.8.1. First Hypothesis

Less than the knowledge required to engineer the software-intensive solution causes distortions in intermediate products that propagate through the sequence of engineering transformations and manifests as defects in the software. Fixing distortions in the software that were caused by lack of knowledge often requires knowledge that is not available in the software development domain and requires additional time and effort. Lack of knowledge is therefore one of the causes of cost overruns and schedule delays and eventual failure of software-intensive acquisitions.

The first hypothesis can be formulated as follows:

Lack of knowledge required to engineer the software-intensive solution increases cost, extends schedule and reduces the effectiveness of the solution.

The term ‘knowledge’ in the hypothesis has a broader meaning and should be understood as ‘knowledge, experience and skills’ required to perform a task (see Section 2.4.7). Although this hypothesis reflects an empirical truth, the history of failure of software-intensive acquisitions demonstrates that the real impact of lack of knowledge is poorly managed and not well understood.

A computer-simulated environment will test the first hypothesis by exploring the absence of essential knowledge in software-intensive acquisitions.

3.8.2. Second Hypothesis

Structures that promote constructive behaviours, as discussed in Section 2.4.6, should facilitate learning behaviours fostering collaboration, information sharing and learning, and create the knowledge to engineer the solution. Conversely, systemic structures that promote aggressive and passive behaviours move away from learning behaviours and should contribute negatively to the performance and effectiveness of the acquisition.

A second hypothesis can be formulated to address the effects caused by the first hypothesis:

Systemic structures that facilitate learning behaviours improve cost and schedule performance and the effectiveness of the solution of software-intensive acquisitions.

Again, the history of failure of software-intensive acquisitions demonstrates that the way structure influences behaviour is not well understood and also poorly managed. Often projects that show problems in cost and schedule are managed in ways that create aggressive-passive behaviours that do not address the real problem, e.g. lack of knowledge, communication and collaboration, and worsen the situation.

A computer-simulated environment will be used to investigate the impact of systemic structures on the success or failure of software intensive acquisitions.

3.8.3. Testing the Hypotheses

To test the hypotheses in a computer-simulated environment the model and respective simulation need to include characteristics that are representative of the real system identified in this chapter.

The simulation will represent the products that are the subject of the acquisition and their inter-dependencies; the engineering process and the life cycle development process; and the knowledge that is required to perform the engineering tasks.

The simulation will capture the way that people execute their tasks in accordance with their abilities, i.e. knowledge, experience and skills. The effectiveness of the final product will reflect how well the task was performed.

People in the simulation will learn, collaborate and interact with each other in accordance with their behavioural style representing their personality. People will adapt by learning and changing behaviour as they interact with each other, with the environment and the product they develop, representing this way the characteristics of a social complex adaptive system.

3.9. Summary

Software-intensive acquisitions are socio-technical systems and complex adaptive systems where people as individuals and in organisations interact to achieve the aim of the system. The system has explicit and implicit aims. The explicit aim of the system is to acquire a solution for a need that relies on software-intensive products, while the implicit aim of the system is to achieve the goals of organisations and individuals. Explicit and implicit goals often conflict.

The software-intensive acquisition process is a sequence of transformations that transform input products into output products. Transformations require knowledge and effort. Lack of knowledge, skills and experience create distortions in intermediate products that propagate into the software like a Chinese Whispers effect. The high level of abstraction that is needed to manage the complexity of software-intensive solutions leaves details to be resolved in the software domain, that if unresolved become distortions. Fixing these distortions in the software requires knowledge of domains that are foreign to software engineers and is one of the causes of increasing cost and schedule in software-intensive acquisitions.

Knowledge is a central aspect of the success of software-intensive acquisitions. Software-intensive solutions ask for knowledge of multiple domains and carry uncertainties created by what is not known, including unknown ‘unknowns’. Knowledge is distributed among people and organisations. Collaborating and sharing existing knowledge are essential to create collective knowledge, and also as part of a process of discovery to reveal the unknown ‘unknowns’. The development life cycle, learning-based processes, learning and constructive behaviours can aid in creating collective knowledge and the process of discovery.

Software-intensive acquisitions fail when there are unrealistic expectations about foreknowledge and effort. The acquisition may fail because of lack of technical knowledge, skills and experience to engineer the solution, and lack of management knowledge to recognise and plan for this deficiency. The acquisition may also fail when unrealistic estimation creates an also unrealistic expectation about cost and duration. The effect known as ‘the mythical man-month’, (Brooks 1995), manifests when the effort of learning and discovering is not considered in the estimation of tasks that cannot be completely defined.

As complex adaptive systems, organisations can develop desirable and undesirable emergent properties. The ultimate desirable property emerges when the acquisition achieves the explicit and implicit aims: the solution satisfies the need and individuals and organisations also achieve their goals. Three also desirable emergent properties (knowledge, competency and quality) work together so that the system achieves its ultimate goal.

The software-intensive acquisition is unable to manage itself and this is the task of management and leadership that should design and foster systemic structures that facilitate constructive and learning behaviours.

The investigation of the software-intensive acquisition system led to two hypotheses, one about the impact of lack of knowledge in the success of the acquisition, and the other about how systemic structures that facilitate learning behaviours can help the acquisition to achieve its goal. The hypotheses will be tested in a computer-simulated environment that is representative of the real software-intensive acquisition system.

Chapter 4: Understanding Variation in the Software-Intensive Acquisition System

4.1. Observed Variation in Software-Intensive Projects

Variation can be observed in the wide range of software development productivity benchmarks, in the low reproducibility and in the rate of success and failure of large software projects. Variation is seen as a deviation in performance, and little is known about why it happens. Why is it difficult to predict how long a software-intensive project will take and how much it will cost? The answer can be found by understanding variation.

As far as software-intensive acquisitions and engineering projects in general are concerned, variations may reduce the expected acquisition performance causing schedule delays, cost overrun and systems that are not adequate for the intended purpose. Conversely, variation can also make the acquisition of a software project perform better than expected, or at least better than the organisation was capable of achieving when the project started. The US Office of the Assistant Secretary of the Navy (Research, Development and Acquisition) recognises that not all variations observed in software development are bad, and ‘some variations might be benign while others might represent an emerging risk item which must be tracked and, if necessary, mitigated’ (USN 2008 p. 9-5). Given the right conditions, variation may occur in more desirable ways and cause the organisation to perform better.

This research is interested in large software projects often required for the development of military and embedded systems. Large software projects have between 10,000 and 100,000 Function Points (FP), roughly developing between 1,000,000 and 10,000,000 source code statements; small projects have less than 1,000 Function Points (Jones 1996). On average large military software projects have staffing levels in the range of 120 to 1,300 staff, for developing respectively 1,000 and 100,000 FP (Jones 1996 Table 3.23, p. 203).

Military projects follow military standards for software development and adopt best practices models like CMM. Project management and quality control activities like reviews, testing and quality assurance, are an attempt to control variation. A study

carried out by Capers Jones reported that ‘good quality control is the best overall indicator of a successful project’ (Jones 2003 p. 27). Standardisation and best practices can reduce variation. However, military projects still present high levels of variation. Military standards and best practices call for an extensive set of activities and a large number of specialists. Large military software projects often perform in the order of 25 software development activities, shown in Table 9, taken from Jones (2008 Table 3.12, p .257).

Table 9 – Software Development Activities for Large Military Projects

	Software Development Activity	Purpose	Average Effort (%)
1.	Requirements	Software Development	7.0
2.	Prototyping	Software Development	2.0
3.	Architecture	Software Development	1.0
4.	Project Plans	Project Management	1.0
5.	Initial Design	Software Development	6.0
6.	Detailed Design	Software Development	7.0
7.	Design Reviews	Quality	1.0
8.	Coding	Software Development	16.0
9.	Reuse Acquisition	Others	2.0
10.	Package Purchase	Others	1.0
11.	Code Inspection	Quality	1.0
12.	Independent Verification and Validation	Quality	1.0
13.	Configuration Management	Support	1.5
14.	Formal Integration	Software Development	1.5
15.	User Documentation	Others	10.0
16.	Unit Testing	Quality	3.0
17.	Function Testing	Quality	5.0
18.	Integration Testing	Quality	5.0
19.	System Testing	Quality	6.0
20.	Field testing	Quality	3.0
21.	Acceptance Testing	Quality	3.0
22.	Independent Testing	Quality	1.0
23.	Quality Assurance	Quality	1.0
24.	Installation and Training	Others	1.0
25.	Project management.	Project Management	13.0

From Table 9, military projects spend 30% of a project’s effort on quality control activities, while project planning and management consumes on average 14%. Just over 41% of the effort is actually spent on software development, while 44% is spent on the

attempt to control variation without any guarantee it will be effective. It is interesting to observe that only 1% of the effort is spent on planning, which carries out the estimation of effort and schedule. This is an indication that variation between planned and actual in the course of the project is not only caused by variation in performance but also by unrealistic estimation and planning.

4.1.1. Observed Variation in Quality

Software products are built to meet a specification. Defects are the discrepancies between the software specification and the software product. A perfect software product would present zero defects and would perform not only as specified but ultimately as required. If there were no variation in the quality of the software there would be no need for rework and there would be no delivery delays and cost overruns. Reality, however, is quite different.

Table 10, taken from Jones (2008 Table 3-47 p. 319), illustrates the number of defects present in the software at the delivery as a function of the project size expressed in Function Points. A military software product with 10,000 function points would contain in the order of 6,200 defects at delivery and about 1,685 of these defects would be critical or significant problems; if the software is ten times larger, i.e. 100,000 function points, the number of high-severity defects present at delivery is expected to be at least 15 times more (Jones 2008 Tables 3-48 & 3-49 pp. 319-320).

Table 10 – Average Number of Delivered Defects per Function Point

Application	Project Size in Function Points		
	1,000	10,000	100,000
Commercial	0.40	0.64	0.92
Military	0.47	0.62	0.77

If a Function Point is an output product of the software development process, Table 10 shows that for large software projects on average 62% of the software products are defective, reaching 77% of defective products for larger projects. A hypothetical comparison with an equivalent manufacturing production line would show that this software production is not a viable business.

4.1.2. Observed Variation in Productivity

Software development productivity is often reported as a number of software product units per staff-month. Function Points is a more acceptable unit, although productivity is still largely reported as the number of programming statements, or source lines of code (SLOC), per staff-month. The number of SLOC required to implement one Function Point is dependent on the programming language. One Function Point corresponds to 60 to 80 (71 average) SLOC of Ada 83 and 40 to 140 (55 average) SLOC of C++. The variation is attributed to individual programming styles and also to unknown causes (Jones 1996).

Productivity in software development depends on many factors. Project size, application domain, team size, schedule, programming language and development methodology are often associated with the measure of productivity. Human factors also impact on software productivity and will be discussed in Section 4.2.

The information available on software development productivity is the result of the analysis of historical data and reflects the performance of software development activities; the information does not take into account the activities prior to software development, as for example the engineering effort to describe the need and to design and specify the system. Deficiencies in these activities are considered to be ‘creeping requirements’ as far as software development productivity is concerned. Table 11, from Jones (2008 Tables 3-17 and 3-18, pp. 269-270), shows the probability of a project achieving a productivity range based on project size and the application domain. The lower probability of achieving the target productivity is an indication of the high variation of software development productivity, which also explains the high variation in cost and schedule experienced by large projects.

Table 11 shows that productivity and its variation are reduced with an increase in project size. Military projects show the same effect when compared with equivalent commercial projects. The reduction of productivity as well as variation can be explained by observing that larger projects often spend more effort on support and quality control processes. The lower productivity and lower variation of military projects when compared with commercial projects is not an intrinsic characteristic of the application domain, but it is a consequence of the higher level of quality control and support processes adopted by military projects.

Table 11 – Productivity Probability Ranges

Productivity (FP/Staff-Month)	Project Size in Function Points			Application	
	1,000	10,000	100,000	Commercial	Military
>100	0.0%	0.0%	0.0%	3.0%	0.0%
75-100	1.0%	0.0%	0.0%	7.0%	0.0%
50-75	2.0%	0.0%	0.0%	10.0%	0.0%
25-50	3.0%	0.0%	0.0%	15.0%	7.0%
15-25	20.0%	2.0%	0.0%	20.0%	12.0%
5-15	55.0%	15.0%	20.0%	27.0%	24.0%
1-5	15.0%	63.0%	35.0%	15.0%	42.0%
<1	4.0%	20.0%	45.0%	2.0%	15.0%

The productivity data in Table 11 reports findings around the year 2008 (Jones 2008). Similar data was published in 1996 (Jones 1996) and some reduction of variation can be observed for both commercial and military projects in the productivity range of 5 to 15 FP/staff month. Such improvement can be attributed to the adoption of quality standards and the Capability Maturity Model (CMM).

Schedule pressure also influences software development productivity. Projects that run on tight and sometimes-unrealistic schedules frequently cause managers to exert excessive pressure on development teams to achieve aggressive schedule deadlines. Predominantly transactional leadership, which is focused on the task and not on people, leads to negative consequences (Thite 1999). There is an optimum point at which management pressure works best and serves as a stimulus rather than unrealistic command and control. According to Jones (1996 Figure 4.6, p. 310), the optimum point will depend on the team and management attitude, but often is located just above what would be considered an average pressure. Thite suggests that predominantly transformational leadership combined with a component of transactional leadership yields better results (Thite 1999).

4.1.3. Observed Variation in Schedule

Software development schedules also show large variation. Table 12, extracted from Jones (2008 Table 3.25, p. 290), shows the schedule variation observed in a wide range of software projects. Small projects, with fewer than 1,000 Function Points, present shorter schedules with a very large variation, explained by the fact that these kinds of

projects seldom follow rigorous standards for software development, and adopt ad hoc processes for estimation, software development and project management. The adoption of rigorous software development processes, conversely, is the reason why larger projects exhibit less schedule variation, although still very high.

Table 12 – Software Schedule Ranges in accordance with Project Size

Project Size (Function Points)	Variation	Project Schedule Range in Calendar Months		
		Minimum	Average	Maximum
1	1333%	0.03	0.16	0.40
10	734%	0.32	1.07	2.35
100	633%	3.00	10.00	19.00
1,000	267%	16.56	26.60	44.16
10,000	283%	29.88	49.80	84.66
100,000	300%	44.28	73.80	132.84

Variation in productivity and scope creeping, often caused by the poor quality of requirements, can explain schedule variation of large projects. The schedule ranges and variations shown in Table 12 are the same reported in 2008 (Jones 2008) and 1996 (Jones 1996), therefore no improvement in schedule variation was observed in this twelve year period.

4.2. Causes of Variation in Software Development

Variation in cost, schedule and quality observed in software development are in fact consequences of other forms of variation. As discussed in Section 2.2.2, variations occur due to common causes or special causes (Deming 2000). To be able to predict the variation in the system, the causes of variation and how these impact the system must be identified. Therefore, the first step is to find what are the sources of variation in the system.

Software is the output product that will present variation as a consequence of the development process which is highly dependent on people and thus subject to people's variation in their abilities to perform tasks that are required for the development of software. It is a known fact that individual and social factors influence software development productivity (Brooks 1995; Humphrey 1988), confirmed by initiatives

such as Peopleware (DeMarco & Lister 1999) and the People Capability Model (SEI 2001).

The US Department of Air Force Software Technology Support Center acknowledges that: ‘there are an enormous variations among the capabilities of software engineers, so too there are enormous variations among the capabilities of software development vendors’ (DAF 2000 p. S-67).

To understand the impact of human variation on the development of software it must be understood what the task entails, what people’s abilities are needed and how these abilities influence the process. The first ability that comes to mind is people’s knowledge and skills in software technology, which software engineers and developers should master, at least in what is needed to transform a software solution into software code.

Software development tasks are subject to schedules and budgets that result from a process of estimation. It is therefore reasonable to assume that people’s knowledge, skills and experience in how to estimate software development tasks are also sources of variation that cause observed variations of cost and schedule.

Software development is a highly abstract and complex task. Before the software comes into existence people need to manipulate mental models that will become a logic structure implemented as software instructions. The design and implementation of software require the manipulation of many interconnected components each performing specific functions in the software solution.

The specification and design of software components is documented, written and in graphic form, thus the accuracy and completeness of documents depends on people’s ability to communicate what is in their minds. The interpretation of software documents is also influenced by the ability of the reader to understand what is in the document.

Software development is a collaborative task. Software engineers and developers need to interact with the customer to understand what the software must do to satisfy their needs; and interact and collaborate with other members of the software development team that together develop the software solution. The inputs to the software development process are outputs of other engineering processes also highly influenced

by people's variation. Variations that come from other processes will influence the variation in the software-intensive acquisition system, discussed further in this chapter.

In summary, a non-exhaustive-list of people's variations in their abilities that influence the development of software includes: mastering the technology of software estimation and development; communicating and handling abstract and complex objects; and team-working skills that are highly dependent on personality and interaction style. These kinds of people variations will cause observed variations in cost, schedule and quality of the software solution.

4.3. A Case of Quasi-Perfect Software

It is possible to reduce variation in software development through a well-defined process and the Space Shuttle Onboard Software Process illustrates what can be considered to be 'quasi-perfect software'. What follows is a compilation of pieces of information gathered from NRC (1993) and Fishman (1996) and a comparison with the average costs and duration of software development in other application domains reported in (Jones 1996).

The Space Shuttle Onboard Software comprises two major components, the Primary Avionics System Software (PASS) and the Backup Flight Software (BFS), with a total of 490,000 lines of code written in High-order Assembly Language (HAL/S), which was developed specially for the Space Shuttle. The specification fills 30 volumes and 40,000 pages. The software development process has four main phases (Requirements Definition, Software Design & Code Development, System Build and Verification and System Performance Verification) with several stages each. About 85% of errors are found and fixed before formal test and 99.9% (hence quasi-perfect) before the software is delivered to NASA. To achieve this level of quality, software requirements are almost pseudo-coded and no room is left for creativity at software design and coding. Highly experienced staff write the software requirements, and the programmers, also very experienced, must write the code exactly as the software requirements prescribe.

The software is upgraded regularly to meet the requirements of each mission and to add new features and capabilities. It is reasonable to assume that the 30 volumes of specification are also updated, requiring a high level of knowledge transfer among people to ascertain where in the 40,000 pages of 30 volumes the update changes have to

be applied. From 1983 to 1991 there were 14 upgrades developing a total of 151,900 lines of new and modified HAL/S code. Each upgrade produces an average of 10,000 lines of code (the smallest produced 1,100 lines and the largest 32,000 lines of code). The average software development cycle for each upgrade is 19 months to develop the software plus 9 months for Mission Preparation.

The Space Shuttle flight software costs US\$100 million per year for the complete software development and assurance process. Independent verification and validation (IV&V) alone costs US\$3.2 million per year. The cost to produce 151,900 lines of code in nine years was US\$900 million, which is equivalent to US\$5,925 per line of HAL/S code.

Cost and Schedule

To allow a comparison between the costs to develop quasi-perfect software and other software, an estimation of the cost per Function Point will be made. Although there is no information available on the number of Function Points implemented in the Space Shuttle Onboard Software, it can be conservatively estimated in the order of 1,000³¹, at the staggering cost of US\$900,000 per Function Point. In comparison with software developed for other applications, shown in Table 13 extracted from Jones (1996 Table 3-40, p. 227), the cost of the Space Shuttle Onboard Software is at least 50 times the cost of military software developed for a large project or two orders of magnitude the cost of equivalent software in size developed for any other application and taking four times the time to develop (26.6 months is the average time to develop 1,000 FP – see Table 12).

Table 13 – Cost Comparison of the Space Shuttle Onboard Software

Project Size (Function Points)	Average Cost (US\$)			Space Shuttle Onboard Software Cost (US\$)
	Commercial	Embedded	Military	
1,000	\$1,920	\$2,587	\$8,336	\$900,000
10,000	\$2,944	\$3,553	\$11,232	
100,000	\$4,092	\$5,897	\$16,161	

³¹ The estimation assumes that HAL/S is a Level 2.0 language, situated between basic Assembly (Level = 1.0) and ‘C’ (Level = 2.5). One Function Point corresponds to 160 lines of code of a Level 2.0 language (Jones 1996 Table 2.15, p. 80). The 151,900 lines of HAL/S would be equivalent to 943 Function Points, approximated to 1,000.

It is pertinent to observe that the average cost per Function Point published in the third edition of Capers Jones book (Jones 2008 Table 3-41 p. 313) is much less and of the order of 30-50% of what was published in the second edition (Jones 1996 Table 3-40 p. 227). Jones does not explain the cost reduction. The comparison was made with the data reported in the second edition because it reflects the same period of the Space Shuttle Onboard Software upgrades.

Variation

This research could not find information about variations in the Space Shuttle Onboard Software development process but they are expected to be small and the process is likely to be stable as it incorporates several mechanisms to reduce variation, as follows:

- a. Variations introduced by the personality and creativity of programmers are reduced by very prescriptive requirements and procedures;
- b. Variations caused by human error are reduced by the use of highly qualified professionals with several years of experience with the (same) process and the (same) software, and existing errors are likely to be detected and corrected by reviews, inspections, developmental tests and IV&V procedures; and
- c. A thorough quality assurance process reduces variations caused by non-compliance with the software development process.

Statistical process control (SPC) has been implemented in one of the many processes of software development for the Space Shuttle Onboard Software Project (Florac, Carleton & Barnard 2000). The software inspection process was selected for the experiment because it was stable and one of the best performance processes in the project also known for producing exceptionally high quality software. The experience was reported as positive and the statistical analysis of the gathered data showed some aspects of the process that required further investigation and possibly improvement.

Effectiveness and Efficiency

The software development process for the Space Shuttle has proved to be effective in achieving the objective of producing high quality software that meets the user needs. The question that remains is whether the process is efficient. Efficiency would tell how much of the process contributes to the software and how much of the process does not

add value to the software and thus is unnecessary waste. Are for example all the quality assurance, reviews and test activities necessary? If the software development process were perfect there would be no errors, therefore all support activities would be deemed unnecessary and the waste would reduce the process efficiency. The same could be said about management: an ideal process would run by itself without the need for managers. However, any process that is people dependent is subject to variations. The process is designed to reduce variations and include support tasks with the objective to find and correct undesirable variations.

The Committee that perform the ‘Assessment of Space Shuttle Flight Software Development Process’ (NRC 1993) considered and rejected the decision to eliminate the IV&V function for it being an essential component of the process to achieve its end goal. The Committee identified issues and made recommendations to improve the process and in areas that impact the process including roles, responsibility and organisational learning.

The efficiency of the Space Shuttle Flight Software Development Process, whether high or low, seems to be what the system needs to be effective (Hitchins 2003). The cost of the effectiveness of a quasi-perfect software development process creates implications for the success of software-intensive acquisitions. If the cost of effectiveness is of that order of magnitude, many software-intensive acquisitions would not be viable.

4.4. Variation in Software-Intensive Acquisitions

Variation in software-intensive acquisitions is present throughout the acquisition process. The causes of variation in software-intensive-acquisitions go beyond the variations in the software development process discussed in Section 4.2. Variations start at the definition of user requirements and continue throughout the development process. The US Department of Air Force Software Technology Support Center acknowledges that: ‘variations in the experience and competence of users makes defining user requirements difficult, particularly in the area of human-computer interface (HCI) design’ (DAF 2000 p. 6-28); and states that ‘for software development contracts, it is axiomatic that the greater the competence of the software development contractor, the greater the probability that the software development project will be successful’ (DAF 2000 p.S-66).

Before a software-intensive acquisition is approved it goes through a process of estimation and risk analysis. In Australia, the approval is a two-pass process (DOD-AU 2006, 2009): the first pass approval is the result of an initial study to determine the viability, cost, schedule, return on investment and risks; the second pass is the formal approval before the contract goes to tender, and acknowledges the findings of a more detailed study that specified the system to be acquired, timing, budget and the organisations that will be invited to submit bids for the tender. The effort spent on preliminary studies is intended to increase the chances of success and reduce the risk of failure of the acquisition of complex systems. However, the risk of troubled projects still exists and they do occur. See for example the delays experienced by Australian software-intensive projects like Wedgetail (Bishop 2007), Super Seasprite Helicopter (Ferguson & Blekin 2008), and Collins Class Submarine (Yule & Wolner 2008).

Management is likely to correlate deviation in performance with staff productivity, error density and rework, poor estimation and scope creep. Little effort is spent to understand what causes variations. Variation should be understood to allow prediction, or at least to provide explanation when the system does not achieve its intended purpose. Better understanding of the system and its variations would provide ways to introduce changes to the system that could improve its intended performance.

4.4.1. Common and Special Causes of Variations

The Software Acquisition Capability Maturity Model (SA-CMM) (SEI 2002) recognises the existence of common and special causes of variations in software acquisitions, discussed in section 2.2.2. The SA-CMM recommends at its second highest level of maturity (Level 4 – Quantitative, focused on quantitative management) the use of *causal analysis*³² to determine the special causes of variation in the software acquisition process and then to implement changes in the process to eliminate special causes of variation or make the process immune to them. At its highest level of maturity (Level 5 – Optimising, focused on continuous process improvement) the SA-CMM recommends the use of *causal analysis* to determine the common causes of variation in the software acquisition process and then to implement changes in the process to correct the effects of common causes of variation.

³² SA-CMM defines causal analysis as ‘the analysis of defects to determine their cause’ (SEI 2002 Appendix A, Glossary of Terms, p. A-3).

Variations in software-intensive acquisitions due to special causes are not an integral part of the acquisition process, and are triggered by unpredictable events that impact the acquisition performance. The nature of special causes of variation is an indication that the application of causal analysis requires experience and strategic imagination. Resource unavailability is an example of a special cause of variation (Leach 2000).

Common causes of variation are intrinsic to the system and are driven by chance in accordance with a probability of distribution. Knowledge about common causes of variations allows prediction of the total variation in the system (Deming 2000). Therefore, the first step to understand the common causes of variations is to understand the system, addressed in Chapter 3 of this thesis.

This research suggests that common causes of variation in software-intensive acquisitions originate in the technical and social components of the system, as discussed in Section 3.7, and are influenced by size and complexity, by human factors and by processes.

Variation caused by Size and Complexity

Size and complexity are attributes of both technical and social systems. ‘Size and complexity go hand in hand. The bigger the application, the more complex it becomes’ (DAF 2000 p. 2-22). The number of physical (hardware) and logical (software and procedures) components determines the size of the technical system; while the size of the social system is associated with the number of people and organisations and their relationships.

Software-intensive systems are growing at an alarming rate reaching the level of ultra-large-scale (ULS) systems (SEI 2006), creating high levels of uncertainty for the development of software-intensive systems of all kinds, particularly in defence (DOD-US 2000; NRC 2007).

Integrated systems often combine multiple application domains and different technologies. A naval helicopter, for example the Super Seasprite SH-2G(A) Helicopter is an integrated system that combined several application domains (radar, sonar, electronic-warfare, weapons, navigation and communications) and technologies (embedded computers, air-born data network and glass displays) (DMO 2006; NOC 2006; Scott, Holdanowicz & Bostock 2004). Another example of an integrated system

is the Australian Project AIR-5416, commonly known as Project Echidna, which provides Electronic Warfare Self Protection (EWSP) to various aircraft platforms (Rogers 2008).

Scale and integration increase with advances in technology that brings more powerful computers, larger data storage capacity and faster communication networks. Complexity rises with the number and diversity of interconnected components in the system. The higher the complexity the more difficult it is to communicate and to understand the system. Technical complexity is also a cause of variation in estimating cost, schedule and resources.

As the size of the technical system increases the social system also increases. The larger are the systems, the more difficult it is to comprehend them. The effects of size and complexity on variations in software-intensive acquisitions are therefore associated with people's ability to deal with complexity.

Variation caused by Human Factors

Variations caused by human factors are originated in the social system and are driven by cognitive, psychological and behavioural factors, discussed in Section 2.4.

The difficulty in defining user requirements as a result of variations in the experience and competence of users reported by the US Department of Air Force Software Technology Support Center (DAF 2000 p. 6-28) suggests that variations in the experience and competence of users could be the cause of amendments in specifications and creeping requirements. Jones reports that the average volume of creeping requirements for military projects is 48% and 63% respectively for projects of 10,000 and 100,000 Function Points (Jones 1996 Table 3.28, p. 210).

To engineer large integrated systems requires knowledge, skills and experience not only in systems and software engineering but also in technologies and in the application domains that are integrated in the system. The high numbers of interconnected components that in their own right require specific domain knowledge create a multi-dimensional space that is beyond comprehension of a single human being. Information and knowledge is thus distributed among people in the system, and to create collective knowledge requires information sharing, cooperation and learning. If the social engineering system is not capable of creating a collective knowledge the technical

system will be communicated with distortions. The level of these distortions will depend on the capacity of the engineering team, the knowledge, skills and experience of its members and their ability to collaborate and learn. Less than what is required will cause variations in the performance of engineering activities.

To manage the development of large integrated systems also requires knowledge, skills and experience not only in management disciplines but also in the nature of the task itself. Deming's System of Profound Knowledge is a good indicator of what managing complex projects entails. Managers should therefore have an adequate understanding of both technical and social systems and their variations to be able to manage and guide them to success. It is therefore reasonable to assert that variations in manager's abilities have a profound impact in the software-intensive acquisition system.

The influence of people variation depends on the complexity of the task and processes. Simple tasks can have highly prescriptive processes that leave little or no room for variations. Complex tasks, conversely, require less prescriptive and flexible processes that need to be shaped or even created as the task is executed. Variation in complex tasks is deeply influenced by people variation.

As people interact, perform their tasks and perceive the environment, their knowledge, experience, motivation and attitudes change, for better or for worse. Positive changes can be influenced by systemic structures that promote learning and constructive behaviours, discussed in Section 2.4.

Attributes of knowledge, skills, experience, motivation, personality and behavioural style are directly related to people's abilities to perform their tasks. It is expected that individuals in the population that constitutes the social system of the acquisition will be different and demonstrate these attributes at various levels. Within the population each of these attributes is spread in accordance with a probability distribution described by a mean value and a variance. A highly experienced and uniform team, for example, would have a high mean value and low variance for the attribute that represents experience. In contrast, a low motivated team would have a low mean value for the attribute that represents motivation. In this case, a low variance in motivation indicates that most of the team members lack motivation, while a high variance would indicate that some members of the team are motivated.

The absence of knowledge of how people's attributes (i.e. knowledge, skills, experience, motivation, personality and behavioural style) are distributed in the population, causes uncertainty in the estimation, planning and overall prediction of how the software-intensive acquisition will perform. The inability to predict is therefore not only associated with people's variations but also with the uncertainty about what the variations are. Variability and uncertainty will be further discussed in Section 4.5.

Variation cause by Processes

A number of processes in the acquisition system can help to reduce or amplify variations. In the following is discussed the influence of tendering and bidding processes, the engineering process and the management process in software-intensive acquisitions.

Tendering and Bidding Processes

The tendering and bidding processes are one of the root causes that can trigger variations in the software-intensive acquisition system. Procurement policies that determine the acquisition of military systems by means of public tendering and competitive bids also favour the lowest cost bid (Jones 2002). Competition among the bidders will push offered price to the lowest possible value to raise the chances of winning the contract.

The difficulty in estimating complex technical systems is accompanied with a high level of uncertainty. It is not possible to know for sure whether the estimation is correct and realistic. Whether the winning bidder will be able to deliver the system in accordance with the contract will depend on the correlation between the estimated cost and schedule and the capacity of the contractor organisation to perform the task. The probability of success of a software development project increases with the level of competence of the software development contractor (DAF 2000 p. S-66). If the estimation is proven to be unrealistic when the project is already in progress, pressures to honour the contract and to maintain the business objectives will cause variations in the engineering and software development processes and consequent variations in cost, schedule and quality.

Technical complexity and variations in the maturity of development processes and the experience of bidding by organisations, leads to the assertion that the variation in commercial bids for software-intensive defence systems is high. The history of

performance of software-intensive acquisitions for defence systems is favourable to this hypothesis. As a consequence the acquiring organisation is likely to receive bids with a wide range of costs and conditions on offer. The decision process to select the winning bid takes into consideration not only cost, schedule and technical aspects of the system, it also considers features of the system being offered and the experience and reputation of the contenders to undertake the job. Cost and political reasons, however, often override all the other aspects and can decide against the bid with higher chances of success, paving the path to high levels of variation and possibly total failure of the acquisition enterprise. This situation is illustrated by the technical evaluation to select the Collins class submarine combat system replacement. The technical assessment recommended the German STN Atlas ISUS 90–55 system. However, a political decision imposed the acquisition of the American Raytheon system, creating a risk to the project as reported by Yule and Woolner: ‘While STN Atlas had a system ready to install, choosing the American solution meant beginning another development project with no more than a hoped-for delivery date’ (Yule & Woolner 2008 p. 305).

The approval process of major capital equipment also causes schedule delays before the project starts. The two-pass approval process adopted by the Australian Defence Materiel Organisation (DMO) has a history of schedule delays. The Australian Strategic Police Institute reported that during 2009–2010 only 10 out of 29 planned acquisitions were approved on schedule, whether on first or second pass (Thomson 2011 p. 103). The report also indicates that delays in the approval process have been identified in previous reports and reflect a trend that is likely to continue.

Engineering Process

The engineering process matches a need with a service or product to satisfy that need (Aslaksen 1996). The process transforms problems into solutions by combining the expertise, experience and creativity of engineers with methods and procedures. The performance of the engineering process is thus dependent on human behaviour.

Engineering processes guide problem-solving tasks and are influenced by the Group Interaction Style (GIS) (Cooke & Szumal 1994), discussed in Section 2.4.6. Cooke and Szumal suggest that tasks that require learning and cooperation are better performed when the group exhibit constructive behaviour instead of passive or aggressive behaviours. The combination of the complexity of the problem and the complexity of

human behaviour makes it difficult to understand, quantify and therefore predict variations in engineering processes.

Engineering processes comprise methods and procedures intended to guide engineering activities. The more detailed and restrictive the procedures are, the less should be the variations in the process. The solution to complex problems, however, cannot be prescribed and it is often guided by principles and not so prescriptive methods. It is thus expected that variations are higher in engineering processes that deal with complex problems.

Management Processes

The System of Profound Knowledge is intended to transform what Deming called the prevailing style of management (Deming 2000 p. 92). For the prevailing style of management Deming refers to the style of management that manages by objectives without understanding the system; sets numeric goals without improving the process; delegates quality to someone else without assuming the responsibility (Deming 2000 Chapter 2). The alternative to the prevailing style of management adopts the guidance of the System of Profound Knowledge that aims to improve the system to achieve the intended goals through a better understanding of the system, its components and variations.

The prevailing style of management described by Deming can be related to management that is focused on cost and schedule without knowing whether the system is capable of performing the task within the imposed conditions. Under this type of management, actions to address cost and schedule problems often apply pressure on staff and impose conditions without addressing the real cause of the problem. These actions are typical of aggressive behaviour.

Inadequate application of stretch strategies can explain the poor results of management that are primarily focused on performance targets. Stretch strategy is often used to encourage innovation and knowledge creation by setting difficult targets that are beyond the current abilities of organisational members (Choo 2010). Choo explains that these targets could be attained only if the necessary knowledge gap is acquired. Therefore, when the main focus of the strategy is in achieving performance targets, it could become stressful and counterproductive. Choo suggests that the primary focus of stretch

strategy should be in motivating the members of the organisation through the challenge of learning and problem solving. However, quite often the main focus is on performance instead of developing a sense of challenge that would motivate the members of the organisation.

The dynamics of group interaction (Cooke & Szumal 1994) suggest that in response to aggressive actions from management the engineering team tend to become passive and less motivated to collaborate and learn. This kind of aggressive-passive behaviour is likely to cause several negative effects in the engineering team such as lowering performance, hiding problems and avoiding discussions that may upset management. The team is likely to follow management instructions without questioning or adding any contribution, and the essential and desirable emergent properties (knowledge, quality and competency), discussed in Section 3.6, will not emerge.

It can be argued that management processes that reflect what Deming refers to as the prevailing style of management are likely develop aggressive-passive behaviours and amplify variations in the system. Conversely, it is expected that management processes that adopt the guidance of the System of Profound Knowledge should have better chances of reducing variations in the system.

4.5. Variability, Uncertainty and Risk

The concepts of variability and uncertainty were introduced in Section 2.2.2. Total uncertainty is the combination of variability and uncertainty, which together reduce our ability to predict the future behaviour of the system (Vose 2000). Vose explains that variability is intrinsic to the system and can be reduced only by changing the system. Variability causes variations in the system. Section 4.4 discussed how variations are present in both social and technical components of software-intensive acquisitions. Uncertainty, according to Vose, is caused by lack of information and knowledge about the system and its variations, and can be reduced through investigation and measurements.

When total uncertainty is low it is possible to estimate, plan and predict future performance of the project. This situation was illustrated in Section 4.3 with the complex but stable and well-defined process of the quasi-perfect software of the Space Shuttle Onboard Software Process.

Uncertainty increases with complexity, size and the life cycle of the project (Aslaksen 1996). As the size and complexity of both technical and social components of software-intensive acquisitions are usually high, and life cycle of projects often extends through several years, it is thus expected that software-intensive acquisitions present a high level of uncertainty.

When dealing with complex systems like software-intensive acquisitions, uncertainty can be reduced but not eliminated. If uncertainty and variability are high, the alternative is to manage total uncertainty as risks.

The Project Management Institute (PMI) in its guide to the Project Management Body of Knowledge (PMBOK) states that:

Project risk has its origins in the uncertainty present in all projects'; and defines risk as '... an uncertain event or condition that, if it occurs, has an effect on at least one project objective', and 'objectives can include scope, schedule, cost and quality. (PMI 2008 Chapter 11)

The US Department of Air Force Software Technology Support Center (DAF 2000 pp. 6-7) indicates that software-intensive acquisitions present risk of failure in four ways that may occur individually or in combination:

- a. *The product that was ordered is not the product the end user wanted.*
- b. *The product does not meet performance requirements (operationally or logically).*
- c. *Actual costs are higher than budgeted.*
- d. *Delivery of the product is too late.*

The risks listed above correspond to the four project objectives (respectively, (a) scope, (b) quality, (c) cost and (d) schedule) identified in the guide to the PMBOK (PMI 2008 Chapter 11). Bohem (1991) has identified that risks of the same nature are also present in software development.

The nature of reported risks and the poor history of success of software-intensive acquisitions suggest that the management of the enterprise is risk-centric and a risk in itself. This research suggests that the cause of widespread risk arises not only from

variations in the system but also from uncertainty about these variations. Without accurate information of what the task entails and whether the capacity of the engineering team is sufficient to perform the task within the established parameters of cost and schedule, it is not possible to estimate, plan and predict the performance of the enterprise. This condition leads to the adoption of a management approach that is risk-centric instead of process-centric. The latter would focus on understanding, guiding and improving the system, while the first is driven by uncertainty and therefore a risk in itself.

The adoption the guidance of Deming's System of Profound Knowledge is a sensible approach to reduce uncertainty and variability in software-intensive acquisitions and a way to transform the prevailing style of management from being risk-centric to become process-centric. The new process-centric management style would then focus on understanding the system and its variations and improving the software-intensive acquisition process to better estimate, plan and achieve overall performance.

4.6. Hypotheses about Variation

After investigating variations present in a software-intensive acquisition system two hypotheses can be formulated in response to the question formulated by the second research sub-problem, stated below.

What are the causes of variations in the software development component of software-intensive acquisitions and how do these variations affect the ability to achieve the established parameters of cost, schedule and quality?

4.6.1. Third Hypothesis

The observed variations of cost, schedule and quality of the solution in software-intensive acquisitions are consequences of other forms of variation. The engineering process of complex and innovative systems cannot be prescriptive to the point of eliminating the effects of variations in people's abilities in performing their tasks. In fact, engineering processes for this kind of system need to be flexible to allow the solution to emerge.

A person's attributes of knowledge, experience, skills, personality and motivation impact their ability to perform the tasks that are needed to engineer the software-

intensive solution. Therefore, it is reasonable to hypothesise that variations in these attributes cause variations in the effectiveness, cost and schedule of software-intensive acquisitions.

The third hypothesis can be formulated as follows:

Variations in people's knowledge, experience, personality and motivation cause variations in the effectiveness, cost and schedule of software-intensive acquisitions.

Testing the third hypothesis will investigate in a computer-simulated environment the impact of variations on people's knowledge, experience, motivation and behavioural styles in the observed variation of cost, schedule and effectiveness of the software-intensive solution.

4.6.2. Fourth Hypothesis

The fourth hypothesis is a consequence of the second hypothesis. If systemic structures that facilitate learning behaviours improve cost and schedule performance, then the effectiveness of the solution of software-intensive acquisitions is because these structures improve knowledge, experience and behaviour. Therefore, it is reasonable to hypothesise that these systemic structures will also reduce people's variations and the perceived variations in the acquisition performance.

The fourth hypothesis can be formulated as follows:

Systemic structures that facilitate learning behaviours reduce people's variations and variations of cost, schedule and the effectiveness of the solution of the software-intensive acquisitions.

The same comments made for the second hypothesis in Section 3.8.2 also apply to the fourth hypothesis. The history of poor performance of software-intensive acquisitions shows that the way structure influences variations is not well understood and poorly managed.

Testing the fourth hypothesis will investigate in a computer-simulated environment the impact of systemic structures on people's variation of knowledge and behaviour and the variation of cost, schedule and effectiveness of the software-intensive solution.

4.6.3. Testing the Hypotheses

The hypotheses about variation will be tested in a computer-simulated environment. Therefore the model and respective simulation need to include characteristics that are representative of variations in the real system.

The simulation will include people with individual attributes defining their roles, knowledge, experience and behavioural style.

4.7. Summary

Large variations in cost, schedule and quality are a fact of software-intensive projects. On average more than 60% of software functions in large military projects present defects, and variations in the schedule reach 300%. Engineering and software development processes can reduce variations to some extent and are more successful when applied to simple and well-defined projects.

The Space Shuttle Onboard Software Process was presented as a successful case of variation reductions through the application of a well-defined process producing quasi-perfect software, achieving extremely low defect rates and very high level of effectiveness. However, the software development for the Space Shuttle took longer and cost 50 times the cost of a large military project and 100 times the cost of a commercial project of the same size. If this were the real cost to develop high quality software with low variation, many software-intensive acquisitions would not be viable.

Schedule delays and cost overruns of large software projects can be associated with variations in software development productivity and scope creep, and both cases can be associated with people's variations. People present variations in their ability to perform engineering and software development tasks, whether for variations in their knowledge, skills and experience or variations in personalities that drives the ability to communicate and collaborate with others to share and create collective knowledge and finding the solution for complex and abstract problems.

The investigation showed that the poor history of success of software-intensive acquisitions could be linked to variability and uncertainty. Variability is associated with variations present in the social and technical components of the software-intensive acquisition system. Uncertainty exists because of lack of knowledge and accurate information about variations in the software-intensive acquisition system. This research

suggests that improving the understanding of variations in the system would transform the management of software-intensive acquisitions from being risk-centric to become process-centric.

The investigation of variations in the software-intensive acquisition system led to two hypotheses: one about the impact of people's variations on the observed variations in cost, schedule and the effectiveness of software-intensive acquisitions; and the other, about how systemic structures that facilitate learning behaviours can reduce people's variations and variations in cost, schedule and the effectiveness of the software-intensive solution. The hypotheses will be tested in a computer-simulated environment that is capable of representing the variations in the software-intensive acquisition system.

Chapter 5: The Software-Intensive Acquisition Model

5.1. A Case for Modelling and Simulation

The history of software-intensive acquisitions and capability development suggests that success or failure is a matter of chance. At the beginning of every project, specifications are written, providers are selected, contracts are signed, plans are made, processes are defined, people are gathered and it seems that everything is set for a success story. Every precaution seems to have been taken. Risk management and contingency plans are put in place. Nobody embarks on such an endeavour thinking that it could fail. However, reality shows that few large software-intensive projects truly succeed in delivering an adequate solution on time and on budget. It seems that there is no approach, method, process or technique that guarantees success. Even experience is not always enough as managers often repeat the same mistakes and make entirely new ones.

When problems arise and rescue plans fail the question about what went wrong is in the air. Most likely something was missing. What could it be? It would be helpful if the project leaders could know in advance what the project is lacking so that more precautions could be taken. Less than success is not acceptable and trial and error is not an option. The best option is to find ways of knowing in advance what can go wrong and what would be the best possible way to act when the project behaves differently from the plan. Management is about prediction. How to improve prediction of what can go wrong with large software-intensive projects?

Many software-intensive acquisitions are one of a kind that do not have time or budget to spend on live experimentation. There is only one chance to hit the target. Software-intensive acquisitions are complex social systems that cannot be sufficiently treated by analysis, scientific methods and applied sciences (Rechtin 2000). Without reliable analytic and predictive tools, designing organisations to achieve specific goals becomes an experimental activity of trial and error, which often presents high risk and prohibitive costs. What would be the best approaches, processes and people required to meet the constraints imposed on the project are, often questions that are not adequately answered. Furthermore, what would be the best course of actions in case resources are not fully available is likely to be unknown.

The investigation of sub-problem 2 revealed that software-intensive acquisitions fit into the complex adaptive system model. Computer modelling and simulation is indicated to study the behaviour of complex adaptive systems (Holland 1995) and can be extended to the learning of human systems and the influence of management in the life of organisations (Stacey 1999). Simulation can be used as a learning environment that allows managers to experiment and try alternative solutions without the drawback of trial and error in real systems. Stacey (1999) suggests that simulation is more adequate to understand the dynamics of human systems than traditional methods that adopt interviews and surveys, which do not reveal what is really happening because people in organisations say one thing when they are doing another and usually do not even know what they are doing or why. Simulation can be free of misleading information and allows researchers to hypothesise and test their hypothesis without interference or interfering with the system. However, often simulation makes simplified assumptions and conclusions and is only as good as the fidelity of the model.

This literature suggests that modelling and simulation is an approach to aid the task of predicting what can happen with the software-intensive acquisition and it is a way to test the acquisition during the planning phase to find what can go wrong and how to avoid or minimise the impact.

The research adopts modelling and computer simulation as ways to test hypotheses and run experiments to formulate theories about software-intensive acquisitions without the risks and costs that a real experiment would impose.

Social systems and organisations are complex adaptive systems. A Complex Adaptive System (CAS) is an aggregate of independent entities with the capacity to modify their behaviour in response to stimuli. When interacting with one another these entities produce collective emergent and unpredictable behaviour (Holland 1995).

This research develops a model of software-intensive acquisitions as a complex adaptive system, where people interact and adapt to achieve a common goal. The model is implemented as a computer simulation and used to explore the likely behaviour of the system under different conditions created as simulated scenarios. The aim of the simulation is to test the hypotheses formulated in Chapters 3 and 4 and develop new insights about what contributes to success or failure of software-intensive acquisitions under conditions of interest.

5.2. Modelling and Simulation applied to Social Systems

A model is a representation of something (e.g. artefacts, entities and systems³³ – collectively referred to here as systems) that would be impractical to be presented in its original form for the purpose of analysis, investigation, testing or experimentation. Models can be used to communicate and facilitate the understanding of complex systems, as well as to experiment with how the system would react under specific conditions and assist in predicting the behaviour of the real system. The folkloric statement, ‘essentially, all models are wrong, but some are useful’³⁴ is often used to express the nature of modelling (Rahmandad & Sterman 2008; Silverman et al. 2004; Sterman 2002). Being a representation, models do not contain all the aspects of the real system, thus they are essentially wrong. However, a model is useful when it provides a representation of the real system that is sufficient to meet the objectives of the model. The art of modelling is therefore about incorporating into the model aspects of the real target system considered to be sufficiently adequate for achieving the intent of the model.

Computational modelling and simulation have long been applied in the field of social sciences as a way to develop an understanding of social systems using computational resources (Gilbert 2008; Miller & Page 2007; North & Macal 2007; Rahmandad & Sterman 2008). People and groups of people in computational social models are referred to as actors or agents. Computational models that objectively express features of the target social systems allow the discovery of things about the target social system by investigating the model through computer simulation (Carley 1999).

Among the various techniques available for computational modelling and simulation, system dynamics (SD) and agent-based modelling (ABM) have been successfully used to model and simulate social systems (Gilbert 2008; Miller & Page 2007; North & Macal 2007; Rahmandad & Sterman 2008). Each technique has its own merits and drawbacks that should be carefully considered when deciding the approach that is most

³³ Artefacts are objects without life or consciousness; entities are living beings with some form of consciousness; systems comprise interconnected entities and artefacts that collectively achieve the purpose of the system.

³⁴ Also attributed to the statistician George E. P. Box (source Wikipedia, http://en.wikipedia.org/wiki/George_E._P._Box).

appropriate for a specific target system and the objectives of the model. The characteristics and application of SD and ABM will be presented further in this chapter.

Models constructed with either SD or ABM can be deterministic or stochastic. Given the same set of conditions, deterministic models will produce the same results each time the model is executed. Stochastic models present non-deterministic behaviour driven by chance and random events, and are likely to produce different results each time the model is executed under the same conditions. Non-determinism is one of the aspects that should be considered for model variation in social systems.

5.2.1. System Dynamics Models

The technique known as System Dynamics, also referred to as equation-based modelling (EBM), was developed by Jay Forrester (1958) as an approach to modelling social systems with the intent to aid decision-making processes. SD models make use of differential equations to express temporal cause-and-effect between interacting variables in the system.

System Dynamics adopts a top-down representation of the system with a high degree of aggregation between its components. Actors that share the same characteristics are modelled as groups that are assumed to be homogeneous and well mixed. The behaviour of the group should reflect the collective behaviour of all individuals that have the same characteristics. System Dynamics models have been successfully used to model social systems with a large number (thousands to millions) of actors where collective behaviour is more important than the behaviour of individual actors (Rahmandad & Sterman 2008). System Dynamics models are not indicated to model social systems where individual behaviour, actor interaction and heterogeneity are important to investigate the social phenomenon of interest (Gilbert 2008; Miller & Page 2007; North & Macal 2007; Rahmandad & Sterman 2008).

5.2.2. Agent-Based Models

Agent-based models adopt a bottom-up approach by modelling the system components and their interactions instead of the system as a whole unit. Agent-based modelling and simulation (ABMS) is an intrinsic computational approach for modelling and simulations that has its roots in the study of complex adaptive systems and its characteristics fit well to model social systems as CAS (Miller & Page 2007; North &

Macal 2007). The components of the system, being actors or artefacts, become agents in the model. The behaviour of the system thus emerges through the interaction of its agents.

Important Features of ABM

Agent-based models should present the following six important features (Fagiolo, Windrum & Moneta 2006; Gilbert (2008), shown below within the context of the software-intensive acquisition system.

Ontological Correspondence

There is a direct correspondence between agents in the model and the real system. In social systems people become cognitive agents, named ‘actors’, and ‘artefacts’ used and created by people become non-cognitive agents, also named artefacts.

Heterogeneous Agents

Each agent in the model should operate in accordance with specific characteristics and rules that reflect their correspondent in the real target system, i.e. actors and artefacts in the model should reflect respectively the behaviour and actions of people and the attributes of artefacts in the real system.

Bound Rationality

Cognitive agents should present rational limitations, as people would have in real life. The model could allow the creation of ‘perfect actors’ for testing ideal conditions.

Agent Interaction

Interaction between agents should also be modelled. It is expected the model would include a variety of interactions representing formal and informal relationships between people and types of tasks that relate people and artefacts.

Learning

Cognitive agents in agent-based models should be able to learn in accordance with their actions, experiences and interaction with other agents. Learning is a fundamental aspect of adaptive agents.

Representation of the Environment

The environment where the agents exist and interact should be included in agent-based models. The environment may be physical, conceptual or abstract. Physical environment represent a physical space where actors move about and meet other actors and artefacts. Conceptual environments are defined by organisational and social structures that allow actors to interact with other actors and artefacts even if they are not physically in the same space. Abstract environments are determined by social and organisational culture and mood. Cognitive agents are influenced and can influence the environment where they exist.

5.2.3. Verification and Validation of Social Computational Models

For the model to be useful it must be verified and validated. Verification is the process of making sure the model does what it was designed to do. The verification of computational models tests the computer program under its design conditions to assert that it operates in accordance with its specification and design. Errors detected during verification that are considered of significance should be corrected.

Validation is the process of making sure that the model adequately represents the target system so that it is useful to meet its objectives. If the model was verified but is not validated it will be of no practical use. The correction of validation errors often requires changes in specification and design of the computer program.

The validation of computational social models is a complicated and controversial process due to issues related to the validation criteria implied by the objectives of the model and the difficulty to obtain suitable data to allow systematic validation (Gilbert 2008). Another factor that may complicate the validation of social models even further is the lacking of undeniable validity of the social theory incorporated into the model (North & Macal 2007). Computational social models may never reach an unquestionable validation, although still be useful.

The approach adopted to validate social models should be consistent with the model's intended objectives that can range from reproducing specific to more broad aspects of the system. For the purpose of validation, Gilbert (2008) classifies social models as abstracts, facsimile and middle-range.

Abstract models represent some basic aspects of the social process without addressing any specific situation. Abstract models are often used to develop or evaluate theories when it may be difficult to find any connection with observable data. The same criteria used to validate a theory, as for example, by observing that the model presents the same macro behaviour as the target social system, can be applied to validate abstract models.

Facsimile models are intended to represent a specific aspect of the target social system with a high level of fidelity as possible with the intention of making accurate predictions. Facsimile models are meant to be validated quantitatively against data obtained from the real system.

Middle-range models intend to reproduce some specific social phenomenon without addressing any specific instance of the social system. Middle-range models can be useful to reproduce aspects of the software-intensive acquisition process without being specific to any particular acquisition program. The validation of middle-range models tends to be qualitative because the generic nature of these models makes difficult a direct comparison with a specific instance of a target social system. Validation should look for similarity between the dynamics observed in the model and in similar instances of the target social system.

5.3. Objectives

The software-intensive acquisition model proposed here is a middle-range model. The objective of the model is to reproduce the behaviour of a software-intensive acquisition social system without being specific to any instance of an acquisition program.

The model is intended to be used as a computer simulation to explore the effectiveness, efficiency, cost and duration of the system under conditions of interest to test the hypotheses formulated in Sections 3.8 and 4.6. The model will focus on the engineering aspects that express the need; define the required capabilities; and, specify and implement the software-intensive solution.

The model and respective simulation will include characteristics that are representative of the real system identified in Sections 3.8.3 and 4.6.3, indicated below:

- a. Engineering tasks, products and their dependencies;
- b. Engineering process and development life cycle;

- c. People that execute and manage engineering tasks in accordance with their abilities defined by individual attributes of role, knowledge, experience, motivation and personality;
- d. Interaction between people defined by formal and informal relationships;
- e. People that adapt by learning and changing behaviour as they interact with each other, with the environment and the product they develop;
- f. The cost and duration of the acquisition and the effectiveness of the final product.

The approach acknowledges the difficulty of obtaining realistic data to instantiate the model and the limitation of social theories. Therefore, the simulation derived from this model is not meant to be a precise and accurate prediction tool. Rather, it should be used as a tool to develop a better understanding about the software-intensive acquisition system. The validation of the model, however, can be achieved by comparing qualitatively the simulation results with what is empirically observed in real software-intensive acquisitions.

5.4. Theory and Hypotheses

Models encapsulate hypotheses derived from theories about the system to be modelled. Hypotheses are founded on scientific evidence, empirical research, observation, educated guesses and anything else that represents what is known about the system. What has been presented in the earlier chapters of this thesis about software-intensive acquisitions, systems theory, complexity theory and human behaviour, is now applied to develop the hypotheses for a software-intensive acquisition model.

5.4.1. Complex Adaptive System Hypothesis

Software-intensive acquisitions are social complex adaptive systems (see Section 3.5) where people associated in teams and organisations perform tasks that contribute to achieve a common goal. People are the source of adaptation, as they change themselves when interacting with each other, their tasks and the environment. People adapt when they learn and respond rationally and emotionally to their interactions. People can improve their knowledge and skills, get motivated, become constructive and willing to cooperate and help others when the social environment is favourable to these changes.

Alternatively, people may become aggressive or passive, less motivated and unwilling to cooperate. Adaptation can produce emergent results that may improve or worsen the performance of the software-intensive acquisition.

5.4.2. Problem-Solving Group Hypothesis

Software-intensive acquisitions are a form of problem-solving group where people work together and apply their knowledge and skills to find a solution for a problem and a way to achieve a common goal. The quality of the solution is therefore dependent on the capacity of the individual and the group.

Problem-solving groups are a dynamic social environment where individual and group performance depends upon how people influence and are influenced by others. The performance of the group thus depends on individual and group interaction style (see Section 2.4.6) classified as constructive, aggressive and passive. Constructive individuals seek to achieve the objectives of the group; are often motivated; are willing to learn, cooperate, help and ask for help; and often encourage members of the group. Aggressive individuals are more likely to be motivated to pursue their personal interests; are willing to learn but not often to cooperate and help others; and are likely to blame and reprimand others when the group is not performing well. Passive individuals are less motivated and willing to interact with members of the group; they are likely to do what they are told without questioning; and often execute their tasks with what they have, without asking help when they could.

The higher are the status, role and responsibility of an individual, the more should be the influence of this individual upon others in the group.

5.4.3. Non-Determinism Hypothesis

Individual behaviour is non-deterministic. Behaviour and interaction styles are only an indication of how an individual could behave. Therefore, individual actions are stochastic events determined by a probability of occurrence based upon the individual behaviour and interaction styles in response to a particular stimulus.

5.4.4. Ideal System Hypothesis

An ideal result is achieved when ideal conditions exist. The ideal system comprises of people that are motivated, willing to cooperate and have the knowledge, experience and skills to perform the task. Time, effort and resources allocated to tasks in the ideal

system are sufficient to execute them and achieve the ideal result. The ideal system will achieve ideal effectiveness under its nominal efficiency (see Section 2.3.1).

It is expected that less than ideal systems should not be able to achieve ideal effectiveness, and the system should spend a nominal effort to complete the task within a nominal time, reflecting its nominal effectiveness and efficiency and confirming that ‘the purpose of a system is what it does’ (see ‘The Purpose of the System’ in Section 2.3.1).

5.5. Software-Intensive Acquisition Agent-Based Model

ABMS is an appropriate approach for the objectives of this research. ABM allows the representation of organisations as complex adaptive systems, where heterogeneous cognitive and adaptive agents interact in accordance with a socio-organisational structure, which corresponds to the characteristics of software-intensive acquisitions. ABM will be used to develop a computation model and simulation of software-intensive acquisition social systems that will be explored using computer simulation with the intent of developing a better understanding of the target software-intensive acquisition system.

Agent-based models can be used to simulate a target social system under conditions of interest. During simulation the ABM is instantiated with parameters that configure the model to reflect the social phenomena that one may wish to explore. The configuration parameters determine the number of actors in the social system and their cognitive profile (knowledge, experience, motivation and behavioural style) and the role of the actors in the social and organisational structure. The parameters can also define socio-organisational structure by creating relationships between the actors. Furthermore, the simulation parameters can also define what and how the actors will do to meet the objectives of the social system. If the social system is a software-intensive acquisition, the actors will apply engineering processes to develop artefacts that will be part of a solution to satisfy a need.

The proposed software-intensive acquisition agent-based model (SIAABM) satisfies the six important characteristics of agent-based models (see Section 5.2.2) as shown in Table 14.

Table 14 – SIAABM Conformance to the Important Characteristics of ABM

Ontological Correspondence	There is a direct correspondence between <i>agents</i> in the model and the real system. Cognitive agents, <i>actors</i> , and non-cognitive agents, <i>artefacts</i> , correspond to people and products in the real system.
Heterogeneous Agents	<i>Actors</i> and artefacts will have their own individual attributes. <i>Actors</i> will behave in accordance with their attributes of role, knowledge, experience, motivation and personality. <i>Artefacts</i> are characterised by attributes that define their function and dependencies in the system.
Bound Rationality	<i>Actors</i> will be modelled as cognitive agents with less than ideal experience and knowledge, thus reflecting what is expected of real actors. Actors can be configured with ideal attributes becoming <i>perfect actors</i> for testing ideal conditions.
Agent Interaction	Interaction between agents will be modelled to reflect how <i>actors</i> interact with other <i>actors</i> and with <i>artefacts</i> in the real target system. Interaction between <i>actors</i> will reflect organisational and social relationships (e.g. boss-to-subordinate, peer-to-peer, friendship or acquaintance) and are to be the channel to transmit information, commands, appreciation, help, reprimand and many other forms of explicit and implicit communication. The interaction between <i>actors</i> and <i>artefacts</i> will represent tasks assigned to the <i>actors</i> as work to be performed.
Learning	<i>Actors</i> will be able to learn in accordance with their actions, experiences and interaction with other <i>actors</i> and <i>artefacts</i> . Learning is an aspect of adaptive agents.
Environment Representation	The conceptual environment defined by organisational and social structures that allow <i>actors</i> to interact with other actors and <i>artefacts</i> will be modelled. Social and organisational culture and mood will be modelled. Actors will be influenced and influence the environment where they exist.

The model of cognitive agents in SIAABM, named actors, will develop an adaptive behaviour, that is, the actor's characteristics of behaviour may change in response to interactions with other actors and with the environment as well as with the actor's

actions and experiences. The model will implement aspects of the Life Styles Inventory (LSI) and the dynamic of group interaction styles discussed in Chapter 2, Section 2.4.4. Interaction, learning and adaptive behaviour are of fundamental importance to model social complex adaptive systems.

5.5.1. ACTS Theory

The SIAABM is founded in the ACTS (Agent, Cognition, Task, Social) theory (Carley & Prietula 1994 p. 56) that proposes that organisations are collections of intelligent agents cognitively restricted, task oriented, and socially situated. The ACTS theory embodies a set of propositions as axioms shown in Table 15 and Table 16.

Cognitive agents in software-intensive acquisitions apply their knowledge and perception of the situation in a social environment to perform the tasks of expressing the need, formulate the problem and its constraints, find and implement satisfactorily a solution that resolves the situation need. The SIAABM conforms to the ACTS theory and implements all its axioms as shown in Tables Table 15 and Table 16. The SIAABM contains three main models: the task model that represents the acquisition and engineering processes; the agent model that includes cognitive agents (actors) and non-cognitive agents (artefacts); and the social model that takes care of connections between agents and the organisational structure.

Variation will be modelled as part of the actor model and in the way the population of actors is generated. Each model will be presented in detail further in this chapter.

Table 15 – SIAABM Conformance with the ACTS Theory Agent Axioms

	ACTS Agent Axioms (Carley & Prietula 1994 p. 60)	SIAABM Conformance
Axiom 1	Organisations are composed of goal-directed, intelligent agents (decision makers) who can learn, communicate, and take action in pursuit of goals.	Actors are grouped in organisations, departments and teams. Actors can learn and communicate with other actors to be able to perform their tasks.
Axiom 2	All goal-directed cognitive deliberation, perception, and communication by agents occur within a physical symbol system architecture that is functionally constrained by natural and physical laws.	Actors can be configured as ideal or less than ideal actors, i.e. with bound rationality and constrained by natural and physical laws.
Axiom 3	Symbolic cognitive architectures and their derivative forms, such as Soar and the Computational Models of Newell ³⁵ , sufficiently describe the mechanism by which a goal-directed agent exhibits intelligence, communication, and learning within the constraints of Axiom 2. Such goals need not be articulated or be articulated by the agent. Further, the goals can be automatically generated or selected by the agent as deliberation ensues.	SIAABM implements its own set of models for agents and connections between agents. Cognitive agents are modelled as actors and non-cognitive agents are modelled as artefacts. The connections model defines connections between actors (interaction), between artefacts (dependency) and between an actor and an artefact (task). Actors pursue their goals in accordance with their tasks and motivation.
ACTS Task Axiom		SIAABM Conformance
Axiom 4	An organisational agent performs one or more tasks in an organisation in order to achieve specific personal, task, and/or social (group or organisational) goals, several of which may simultaneously arise and, perhaps, conflict with each other.	Actors perform tasks based on their roles in the organisation, their knowledge, experience, behaviour profile, workload and motivation.
ACTS Social Axiom		SIAABM Conformance
Axiom 5	An organisational agent occupies a position (formal or informal) in the organisation that is a socio-cultural-historical artefact involving one (or more) socially situated roles.	Actors have a formal role in the organisational structure (e.g. Senior manager, Manager, Team Leader, Expert or Team Member) and can develop informal relationships with other actors.

³⁵ Newell et al. 1990; Yost & Newell 1989.

Table 16 – SIAABM Conformance with the ACTS Theory Interlink Axioms

	ACTS Interlink Axioms (Carley & Prietula 1994 p. 60)	SIAABM Conformance
Axiom 6	The agent's current task and social situation define a limited set of knowledge, opportunities, and consequent constraints on actions and learning. Individual deliberation and choice are restricted by the constraints of the architecture and knowledge (the latter of which is affected by personal history, current task, and current social situation).	Actors are constrained by the organisational structure, their knowledge and experience and the opportunities to interact with other actors.
Axiom 7	An organisational agent will select goals and actions enabled by the current task and social situation as perceived (and represented) by the agent.	Actors perform only the tasks assigned to them in accordance with their roles in the hierarchy of the organisation. Actors may take opportunities to interact with other actors created by the organisational structure and constrained by their own behavioural style and motivation.
Axiom 8	Organisations are faced with dynamic environments that can alter the characteristic of the task or social situation temporarily or permanently. Difficulties can and frequently do arise, upsetting the normal or standard operating conditions in the organisation. These difficulties can be with the agents (e.g., lack of experience or poor motivation), the tasks (e.g., change in design and specifications), or the social situation (e.g., turnover level or broken communication channels).	The model creates a dynamic environment that can alter the characteristics of the task and the social environment. Tasks that are performed by less than perfect actors will contain errors that create unplanned work and upset the desirable operating conditions of the organisation. This situation can alter the behaviour and performance of the actors and the quality of their tasks.
Axiom 9	The social situation is continually constructed as individuals engage in concurrent actions. The existence of socially situated positions and all their attendant characteristics (opportunity for interactions and mobility, relative power, links with other positions, status, associated procedures, etc.) is the result of previous interactions among individuals and their decisions and actions. Further interactions and decision making by individuals will alter the social situation and these positions.	Actors decide when and to whom they interact in accordance with the formal and informal structures and the history of previous interactions. Actors will attempt to interact with actors from whom they can get help when needed and will avoid aggressive actors that have previously refused to help. Interactions that produce positive results e.g., obtain help, can influence that actor to improve their knowledge and experience and the overall performance of the organisation.

In accordance with the ACTS theory, the SIAABM comprises the cognitive agent or actor model, the task model and the social environment model. The task model encompasses the non-cognitive agent or artefact model. The social environment model includes the connections that define the social structure.

5.6. Task Model

The ACTS theory sets the context for the task model. In the acquisition of complex systems, people apply their knowledge, skills and perceptions of the situation in a social environment to perform the task of expressing the need, formulating the problem, establishing constraints and finding and implement a solution that satisfactorily meets the need. The original refereed paper published by this research proposing this Task Model is included in Appendix B.

The task model comprises products and tasks and implements the software-intensive acquisition process discussed in Chapter 3, Section 3.3. Products are functional or physical objects that together constitute the whole. Products are a collection of artefacts that can be also functional or physical. Tasks are the transformations applied to products and to produce other products within the same or in other domains. The execution of transformation tasks requires a nominal level of knowledge and skills associated with input and output products and their respective domains.

A nominal effort is associated with the execution of a task. If an ideal agent applies less than the nominal effort, the distortions on the output product will be proportional to the ratio between the actual and nominal task effort.

Although the example adopts linear transformations, the system is not expected to be linear. Software-intensive acquisitions are complex adaptive systems and as people learn and adapt, knowledge and skills are not constant throughout the course of the acquisition. Learning and collaboration can improve useful knowledge. Learning and the application of available knowledge and skills are fuelled by motivation. Divergent motivation can produce non-linear unwanted effects, which in turn creates tension between engineering management and project management, discussed in Section 2.4.2, and are likely to worsen the effectiveness of the system.

5.6.1. Virtual Products and Tasks in the Acquisition Space

In a real acquisition, products and tasks are defined by their functional and physical attributes in the form of requirements, specifications, conditions of operation and processes, expressed in ways to be manipulated by people: that is, textually, graphically and verbally.

The proposed model is intended for simulations in a computational environment, where virtual agents, representing people, teams and organisations, manipulate virtual products and tasks. It would be difficult, if possible at all, to implement agents that would be able to discuss, analyse, write, interpret, implement and verify textual requirements. For that reason, products and tasks will be represented numerically.

Products will be defined in a functional hierarchy and expressed in numerical form. The Analytic Hierarchy Process (Saaty 2000) is used to determine the numeric values based on the contribution of each product to the whole.

Tasks transform products from one level to other products in the next level down. The execution of the transformation tasks requires a nominal level of human resources in the form of knowledge, skills and effort. The nominal knowledge and skills depend on the parent product and its derived sub-products, and effort determines the duration of the task executed by a single agent that possesses at least the nominal knowledge and skills. When an ideal agent applies the nominal effort, the task is executed without distortions and the product is correctly produced, otherwise distortions occur.

Analytic Hierarchy Process

The Analytic Hierarchy Process (AHP) is a framework of multi-valued logic based on the innate human ability to use information and experience to construct ratio scales through paired comparison (Saaty 2000). The object of the analysis is arranged in an hierarchic network structure that breaks down the whole into its smaller parts thus allowing paired comparison. Paired comparison is done using ‘The Fundamental Scale’ of nine levels 1–9, indicated in Table 17 (from Saaty 2000 Table 3.1, p. 73).

The objective of the AHP is to compare all components in the system of interest to determine the weight of importance or contribution of each component to the whole. Although AHP has formal mathematical foundation, included in Chapter 2 of Saaty (2000), it is simple to use, and algebraic calculations are easily performed with the aid

of mathematical software tools. In brief, Saaty's AHP shows that if the system of interest has 'n' components, the pair comparison constitutes an $n \times n$ square matrix named Priority Matrix. The weight of importance of each of the 'n' components is given by the normalised principal eigenvector of the Priority Matrix, which is the eigenvector from the maximum eigenvalue. AHP will be applied to estimate the relative contribution of each of the operational roles of a naval helicopter based on the specification of the Super Seasprite Helicopter, which is used to configure the task model for simulations performed by this research (see Section 6.2).

Table 17 – The Fundamental Scale of AHP

Intensity of Importance	Definition	Explanation
1	Equal importance	The two components contribute equally to the objective
2	Weak importance	
3	Moderate importance	Experience and judgement slightly favour one component over the other
4	Moderate plus importance	
5	Strong importance	Experience and judgement strongly favour one component over the other
6	Strong plus importance	
7	Very strong importance	One component is favoured very strongly over another; its dominance demonstrated in practice
8	Very, very strong importance	
9	Extreme importance	The evidence favouring one component over another is of the highest possible order of affirmation

The Acquisition as Vectors in a Vector Space

Products are modelled as vectors in a Euclidean Vector Space, while tasks are transformations that create other products as vectors within the same or into other vector spaces. Each artefact of the product corresponds to a dimension or a component of the product vector. The numeric value of each artefact represents its contribution to the whole product. The complexity of a product and its artefacts is not determined by the absolute value of its numeric representation, but it is associated with its relationship with other products and by the knowledge, skills and effort required to produce the product.

The number of linearly independent artefacts required to represent the acquisition determines the dimension of the acquisition space. The acquisition space adopts the standard base, comprised of unitary orthogonal vectors corresponding to the dimensions of the acquisition space.

Products are created by transforming an input product P_i by a transformation matrix T_i , as indicated by equation (5.1):

$$(5.1) \quad P_{i+1} = P_i \cdot T_i$$

An acquisition with n products is represented by equation (5.2):

$$(5.2) \quad P_n = P_1 \cdot \prod_{i=1}^{n-1} T_i$$

P_1 is the ‘need’ and P_n is the ‘solution’ product that will satisfy the ‘need’.

Each element of a transformation matrix is a task that requires a nominal knowledge and effort to be executed. The knowledge required contains components of input and output products. If the person, or virtual agent, assigned to the task possesses at least the nominal knowledge on input and output products and applies the nominal effort required, the task is executed without distortions. Otherwise, the task is executed with distortions that are proportional to the lack of knowledge and may take longer if the person decides to apply effort to learn and acquire the required knowledge.

Products and artefacts are transformed into other products and artefacts through tasks executed by the cognitive agents or actors. Artefacts are modelled as non-cognitive agents. Tasks are modelled as linear transformation matrices, plus the knowledge and effort required to perform the task.

Useful Knowledge

The effectiveness of the execution of the tasks depends on the agent’s useful knowledge and is proportional to the projection of the agent’s knowledge vector onto the task’s knowledge vector, shown in Figure 3. The normalised value of 1.0 represents the required knowledge/skills. Although the concept can be applied to an n -dimensional space where each dimension is a dimension of knowledge, the model adopts two dimensions of knowledge. This simplification is consistent with the task model that transforms one product into another. Therefore the two domains of knowledge represent the required knowledge about the domain of the input product and the domain of the output product.

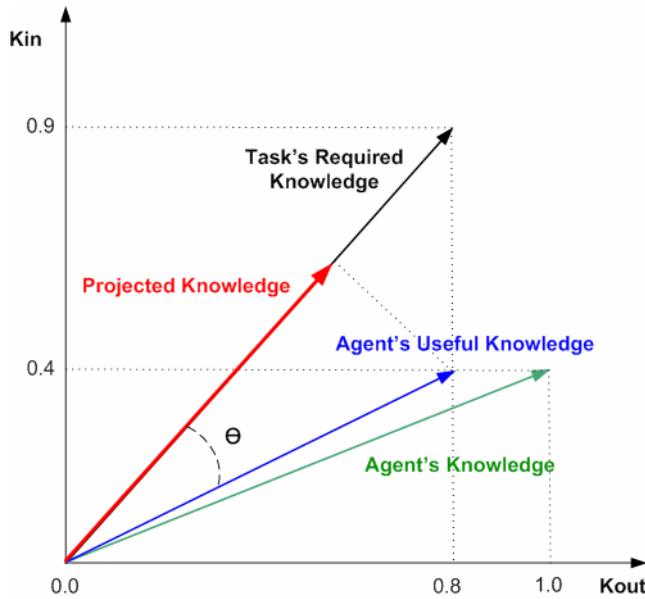


Figure 3 – Useful Knowledge

Effectiveness = $|Agent's\ Useful\ Knowledge| / \cos\theta$, where $\theta = \arccos\left(\frac{ab}{|a||b|}\right)$ and θ is the angle between the two vectors, Task's Required Knowledge and Agent's Useful Knowledge.

The agent's useful knowledge is limited by the amount of knowledge required by the task on each dimension; otherwise the excess of knowledge/skills in one dimension would compensate for the lack of knowledge/skills in another dimension. The effectiveness of the task execution is proportional to the projection of the agent's useful knowledge onto the task's knowledge/skills required.

Effectiveness and Efficiency

The effectiveness of the solution is given by the sum of the contribution of the end products, which in the ideal case is 1.0. Equation (5.3) represents the effectiveness of the solution, where in this example p_i are the m components or artefacts of the final product:

$$(5.3) \quad Effectiveness = \sum_{i=1}^{i=m} p_i$$

Unless distortions are reduced, whether at the end of the development cycle in the form of testing and rework, or as an integrant part of the engineering process, 100% effectiveness cannot be attained.

Equation (5.4) represents the efficiency of the system as the ratio between the productive and actual effort. Productive effort is the effort spent to develop products that are delivered to the customer, while the actual effort includes management and enabling tasks.

$$(5.4) \quad Efficiency = \frac{\sum_{n=1}^{n=p} PEffort(P_n)}{\sum_{n=1}^{n=p} AEffort(P_n)}$$

5.6.2. Chinese Whispers Effect

The proposed task model represents the ideal products, artefacts and transformations found in software-intensive acquisitions. Products and their artefacts are represented numerically by their respective contribution to the whole estimated using the AHP. Tasks are matrices that transform input products into output products, and require a nominal level of knowledge, skills and effort. If less than the required knowledge, skills and effort are applied, output products will contain distortions and omissions. Output products are input products of subsequent transformation and distortions propagate in the form of a Chinese Whispers effect.

Lack of knowledge and skills, can be associated with the failure of software-intensive acquisitions. Knowledge is acquired and shared through learning and collaboration, and these are driven by motivation. Software-intensive acquisitions are complex adaptive systems, and the factors that influence knowledge, are affected by the system and vary during the course of the acquisition.

5.7. Cognitive Agent Model

The cognitive agent or actor model represents the people that are associated in teams and organisations participating in the engineering process. The actor model encompasses all elements of cognitive agents of the ACTS theory. The model implements cognitive agents that pursue their goals of executing tasks to resolve a problem within a problem space, determined by the task definition and the actor's abilities.

Two architectures were considered for the SIAABM actor model: Unified Architecture of Behavior (UAB) (Silverman et al. 2004) and PECS³⁶ Agent (Schmidt 2000), which are not fundamentally different apart from the terminology used. In both architectures agents interact with the environment and other agents through stimuli and responses. UAB and PECS have four internal modules. Three modules model individual characteristics of the agent (cognition, personality/emotion and biology/physics/stress) and one is dedicated to social aspects (social status and relationships). The PECS Behaviour module corresponds to UAB Memory module that determines the agent's actions in accordance with its four internal modules.

UAB adopts the concept performance moderator functions (PMF) that influence the way the agent behaves, while PECS uses state variables. Connections between the various internal models indicate the degree of dependency between PMFs. Basically, every PMF influences and is influenced by all the other PMFs in the model. The integration of multiple PMFs is one of the difficulties in validating cognitive agent models because there is no sufficient scientific evidence about how multiple factors influence one another (Silverman et al. 2004). UAB and PECS respectively suggest that PMF and state variables can be modelled as simple linear functions. Non-linearities emerge due to the integration and interdependency of PMFs or state variables.

The SIAABM actor model is shown in Figure 4 and it is an adaptation of the UAB and PECS Agent models. The SIAABM actor interacts with the environment and other actors through stimuli and responses processed by the Perception and Expression models. SIAABM also has four internal modules: Cognition, Emotion, Physics and Social. The Behavioural model determines the actions performed by the actor. SIAABM introduces a new model, Task Assignment, which is not present in either UAB or PECS. All SIAABM models will be described further in this chapter.

³⁶ PECS stands for Physics, Emotions, Cognition and Social Environment.

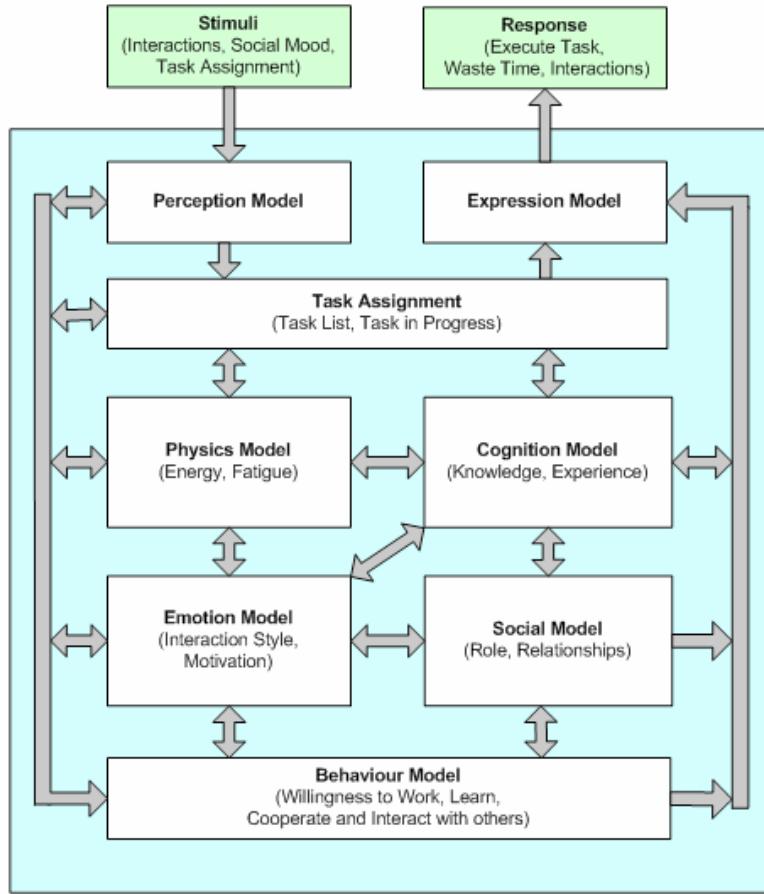


Figure 4 – Cognitive Agent Architecture

5.7.1. Perception and Expression Models

The perception and expression models are the interface between the actor and other actors, the environment and the artefacts assigned to the actor. Actors interact with other actors through communications that can assign tasks; ask and offer help; praise and reprimand, and the like. The actors respond to these stimuli by interacting with other actors, working on tasks and by changing their own attributes of behavioural style, motivation, knowledge and experience. The way that the actors respond to stimuli is not deterministic, but will be determined by a probability of occurrence in accordance with their own attributes.

5.7.2. Social Model

Actors have a *role profile*, shown in Table 18. The role of the actor defines the actor's responsibilities in the organisation. The social structure is defined by roles and interactions. Actors are associated with a team, a department and an organisation.

Teams are led by a team leader; departments are led by a manager; and the organisation is led by a senior manager. The actors keep a list of actors with whom they can interact, including remote actors that may not belong to its team, department and organisation. Each interaction holds a quality factor that indicates the response the actor received when trying to interact with the actor associated with that interaction. Actors that are seeking help will prefer to interact with actors that have higher interaction quality. Actors may also interact with other actors to offer help, praise and reprimand. Interactions are the way actors cooperate and share knowledge.

Table 18 – Actor Role Profile

Role	Description
Senior Manager	Allocates projects to teams, reports progress and aims to maximise profit (provider), reduce costs (customer) or get the best quality (user).
Manager	Allocates tasks to teams, co-ordinates teams and reports progress.
Team Leader	Allocates tasks, co-ordinates team members and reports progress.
Expert	Executes assigned review tasks and provides help to other actors.
Team Member	Executes assigned explicit tasks and may also execute also implicit tasks.

The social structure is defined by simulation configuration parameters, allowing the creation of multiple organisations, departments and teams and defining what role each organisation plays in the acquisition.

SIAABM allows the model to be expanded to address new areas of interest, as for example the influence of internal and external stakeholders. Actors could have a *stake profile* defining the interests of the actor in the outcomes of the acquisition, which in turn would influence the way the actors behave. The actor's stake profile would be influenced by the actor's role and by the organisation to which the actor belongs. As a user, the actor's interest could be to obtain the best solution as quickly as possible, regardless of cost. Actors in management role in the customer organisation would be interested in meeting cost, schedule and quality as specified in the contract. Actors in management role in the provider organisation would be interested in maximising profit and customer's satisfaction. Actors that perform a technical role would likely to be interested in taking the most of the technical challenge and to develop the best technical solution without being constrained by cost and schedule. Observers are actors that do

not participate directly in the acquisition process but having the capacity to influence the final outcome. The stake profile and the influence of observer actors could be driven by a number of factors, including personal and political interests. Although observer actors are not currently included in the model, the architecture of SIAABM permits the creation of new types of actors.

5.7.3. Task Assignment

Actors assign tasks to other actors based on their roles. In an hierarchical organisational structure, tasks are assigned by the actor that has the immediate superior role, i.e., senior managers assign tasks to managers; managers assign tasks to team leaders and these assign tasks to team members and experts. The tasks assigned to an actor are stored in the actor's task list and the actor will work on assigned tasks on a first-in-first-served basis. A real actor may overrule the execution of tasks as first-in-first-served in practice based on preference criteria; however first-in-first-serve is a reasonable assumption for the model. When a task is complete the actor will look for the next task in the list.

The model includes three basic types of task: develop artefact; review artefact; and rework artefact. Only team members and expert actors work productively on tasks associated directly with an artefact. Senior managers, managers and team leaders assign tasks and monitor progress. The main activity of expert actors is to help other actors, but they can also perform review tasks.

The actor's knowledge and experience determine the quality of the tasks it performs and the degree of error in the artefact associated with the task. The aim of review tasks is to discover these errors. The number of errors discovered will also depend on the actor's abilities to perform the task. If the actor does not possess the required knowledge and experience to perform the review, the review task will not discover all existing errors. A real actor may not find all errors even when he or she possesses the required knowledge; however the assumption described is reasonable for the model.

Rework tasks attempt to correct the errors discovered during review. The rework may not correct all existing errors if the actors that performed the review and the rework do not have the required knowledge and experience.

5.7.4. Emotion Model

The emotional model uses the Life Styles Inventory (LSI) (see Section 2.4.4) as the basis to determine the way that actors behave and interact with other actors. The model defines six behavioural styles: aggressive, doer, constructive, nice, passive and neutral, shown in Figure 5. The behavioural style, notated as b , is represented by a normalised numeric value between 0.0 and 1.0 that corresponds to the angle around a circle, and has three components: aggressive a , constructive c and passive p .

Equations (5.5), (5.6) and (5.7) show how the three components of behaviour are calculated.

$$(5.5) \quad \begin{aligned} & \text{for}(0.0 < b \leq 0.33) \\ & \{ p = 0.0, \\ & \quad a = 0.5 * (1.0 + \cos(b * 3\pi)), \\ & \quad c = 0.5 * (1.0 - \cos(b * 3\pi)) \} \end{aligned}$$

$$(5.6) \quad \begin{aligned} & \text{for}(0.33 < b \leq 0.66) \\ & \{ a = 0.0, \\ & \quad p = 0.5 * (1.0 + \cos(b * 3\pi)), \\ & \quad c = 0.5 * (1.0 - \cos(b * 3\pi)) \} \end{aligned}$$

$$(5.7) \quad \begin{aligned} & \text{for}(0.66 < b \leq 1.0) \\ & \{ c = 0.0, \\ & \quad p = 0.5 * (1.0 + \cos(b * 3\pi)), \\ & \quad a = 0.5 * (1.0 - \cos(b * 3\pi)) \} \end{aligned}$$

Pure aggressive behaviour corresponds to 0.0, while pure constructive and passive behaviours correspond respectively to 0.33 and 0.66, as shown in Figure 5.

The actor is created with a behavioural style that may change as the actor interacts with other actors and the environment. The actor behavioural style determines the probability of how the actor will act and respond to stimuli. It is also expected that managers, team leaders and team members will behave differently in accordance with their roles and responsibilities. Equation (5.8) shows that the probability of an actor acting in a certain way is a function of its role and its behavioural style.

$$(5.8) \quad P(\text{action}) = f(\text{Role}, \text{BehaviourStyle})$$

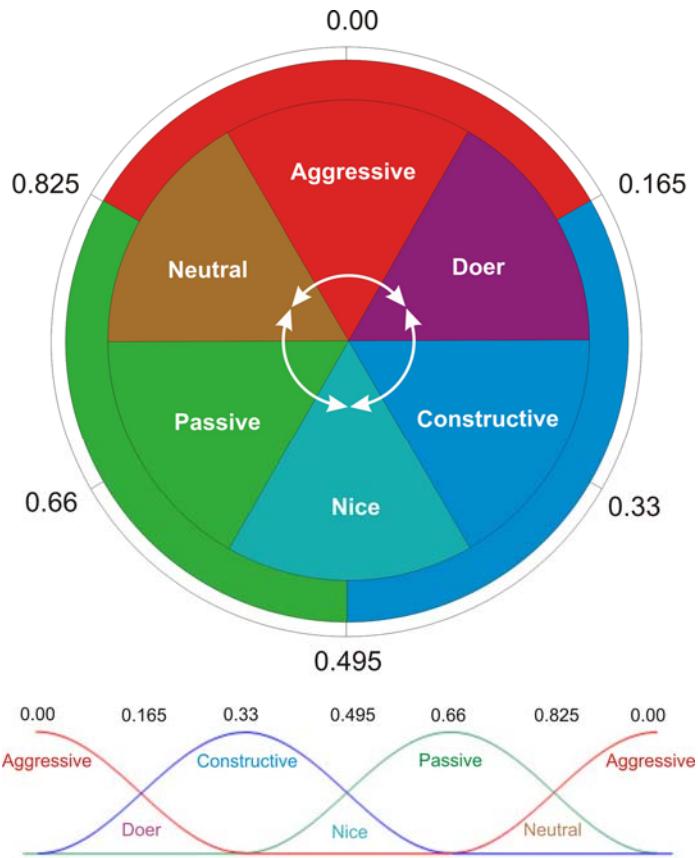


Figure 5 – Actor Behavioural Styles

The various actions and their probabilities of occurrence are defined as simulation configuration parameters as shown in Table 19. The table is constructed by assigning the expected probability of an actor's act in some way in accordance with the actor's behavioural style. The probability table is provided as configuration parameters for the simulation, specified based on empirical data or an educated guess. It is expected, for example, that the probability of a constructive actor to collaborate with other actors is higher than of a passive actor; and that an aggressive actor has a higher probability to reprimand another actor than to offer help or praise.

Table 19 shows the probability of a generic actor to perform various actions. A similar table for each actor role would be needed to express the probability of an event as a function of the actor's role and behavioural style.

Table 19 – Probability of Action in accordance with the Actor’s Behavioural Style

Action	Behavioural Style					
	Aggressive	Doer	Constructive	Nice	Passive	Neutral
Start Interaction	P1a	P2a	P3a	P4a	P5a	P6a
Respond to Interaction	P1b	P2b	P3b	P4b	P5b	P6b
Ask Help	P1c	P2c	P3c	P4c	P5c	P6c
Offer Help	P1d	P2d	P3d	P4d	P5d	P6d
Accept Help	P1e	P2e	P3e	P4e	P5e	P6e
Praise	P1f	P2f	P3f	P4f	P5f	P6f
Reprimand	P1g	P2g	P3g	P4g	P5g	P6g
Learning	P1h	P2h	P3h	P4h	P5h	P6h
Forgetting	P1i	P2i	P3i	P4i	P5i	P6i
Increase Motivation	P1j	P2j	P3j	P4j	P5j	P6j
Decrease Motivation	P1k	P2k	P3k	P4k	P5k	P6k
Working	P1l	P2l	P3l	P4l	P5l	P6l
Work Overtime	P1m	P2m	P3m	P4m	P5m	P6m
Become Aggressive	P1n	P2n	P3n	P4n	P5n	P6n
Become Constructive	P1o	P2o	P3o	P4o	P5o	P6o
Become Passive	P1p	P2p	P3p	P4p	P5p	P6p

In addition to the probability of the actors modifying their own behavioural style, the model also includes a random factor configured as a simulation parameter that may cause the actor’s behaviour to change. The possibility of an actor changing behaviour is driven by chance and introduces random variations into the system that may cause variations in the quality of products as well as in cost and schedule.

5.7.5. Cognition Model

The cognition model represents the actor’s knowledge and experience. The actor’s specialty defines the actor’s knowledge profile. A domain expert actor has knowledge in dimensions associated with the situation domain. The actor can also be knowledgeable on dimensions of multiples domains. A systems engineer actor, for instance, is expected to be knowledgeable in the problem domain and may possess knowledge on dimensions of the situation and solution domains. Table 20 describes the various actor specialties.

Knowledge and experience are normalised numeric values ranging from 0.0 to 1.0. The value of 1.0 represents ideal knowledge or experience. Experience is what the actor uses to apply its knowledge from one domain into another. The model uses the actor’s experience as a factor that converts the actor’s knowledge when applied into another domain. Considering that a number of adjacent domains of expertise (e.g., situation, problem and solution domains), equation (5.9) indicates that the actor’s knowledge K_i about domain i can be applied into domain $i+n$ with a correction factor of E^n , where E is

the actor's experience. If the actor has ideal experience, i.e. 1.0, the actor's knowledge in one domain can be applied into any other domain without any loss.

$$(5.9) \quad K_{d(i+n)} = K_{di} * E^n$$

Table 20 – Actor Specialty Profile

Specialty	Domain Expert	The actor possesses knowledge of a particular domain.
	Systems Engineering	The actor possesses knowledge of the problem and solution domains.
	Software Engineering	The actor possesses knowledge of the software component in the solution domain.
	Test Engineering	The actor possesses knowledge of the testing the solution.
	Management	The actor possesses knowledge of management.
	Support	The actor does not possess specific knowledge.

An ideal actor possesses perfect knowledge in all domains, while realistic actors would not. Realistic actors are likely to be more knowledgeable about their own domain of expertise, although they are able to apply their knowledge into other domains in accordance with their experience as per equation (5.9). An alternative to equation (5.9) is to specify the actor's knowledge for each domain. Figure 6 shows the expected knowledge profile of actors in the situation, problem and solution domains. The knowledge profile indicates that co-operation between actors must occur to share and transfer knowledge.

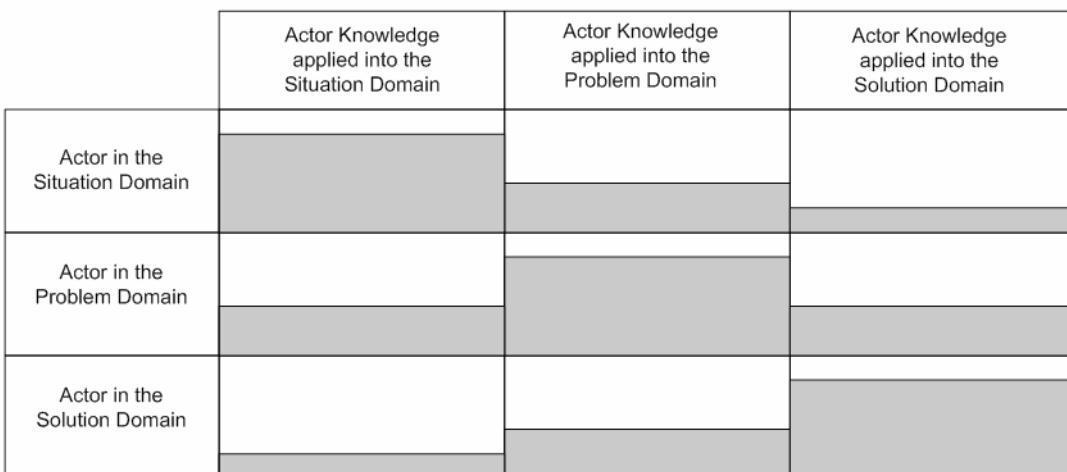


Figure 6 – Actor's Knowledge Profile.

The knowledge profile, or knowledge and experience, is the mechanism to model the distributed nature of knowledge and contains the information that will determine the actor's technical performance.

A task requires specific knowledge and skills to be executed, represented by a vector. A vector also represents the actor's knowledge and skills assigned to the task. The effectiveness of the execution of the tasks will be proportional to the projection of the actor's knowledge/skill vector into the task's knowledge/skill vector, discussed earlier in this chapter (see Useful Knowledge and Figure 3 in Section 5.6.1).

5.7.6. Physics Model

The physics model represents the actor's energy and fatigue, which impact productivity and the quality of work. The actor can work normal hours or more. The probability of how much the actor is willing to work is determined by the actor's role and behavioural style. When the actor works more than normal hours it is expected that its productivity will increase for a limited period. After that the productivity decreases and the quality of the work decreases. When the actor shows signs of fatigue, its productivity drops to levels lower than normal and the tasks it performed will contain more errors. The reduction in productivity and quality is proportional to the fatigue level.

5.7.7. Behavioural Model

The behavioural model combines the effects of the internal models in determining the actor's willingness to work, learn, collaborate and interact with other actors.

The actor's abilities comprise knowledge and experience and can improve through interaction with other actors and with the history of tasks previously executed. The actor's performance will depend on the actor's abilities, rational behaviour (the '*K-factor*') and emotional behaviour (the '*E-factor*') influenced by the socio-organisational situation. Depending on their abilities and motivation, the actors may execute implicit tasks and identify distortions in the input task. Figure 7 shows the relationship between individual attributes in the actor's behaviour.

If the actors are motivated and their abilities match what is required to execute the task, the actors complete their task without errors and within the given time. If the actor's abilities are less than what is required to execute the task, a decision about what to do

will depend on the actor's motivation, determined by role and responsibilities, behavioural style, and the social situation. The actor's options could be:

- Complete the task using its abilities, although the task may be complete with errors;
- Spend time to acquire the knowledge required to reduce the errors, although it may take longer to complete the task;
- Ask for help from another actor to acquire the knowledge required to reduce the errors, although this will take more time, and that of the other actor. This option constitutes co-operation and both actors must be willing to co-operate.

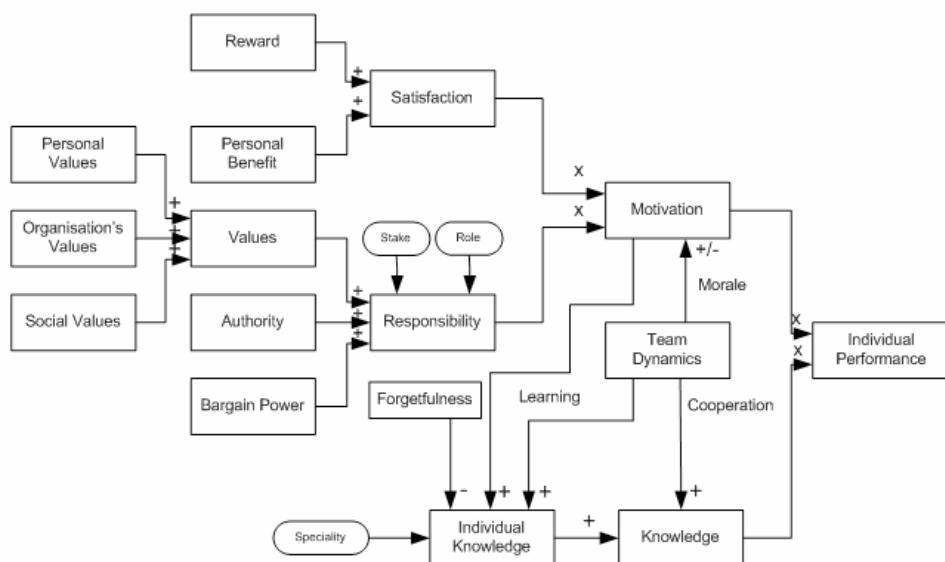


Figure 7 – Actor's Behavioural Model

The actor may also decide to execute implicit tasks, knowing that these tasks are not planned, or may delay explicit tasks and require additional personal effort. Implicit tasks may reveal errors and distortions, which if reported become additional explicit tasks.

The actor's performance depends on the actor's abilities and motivation. The task definition included data (a component of the task vector), specification (a component of the transformation matrix) and the expected effort (duration). Explicit tasks are executed and the result is placed in the Task Out-Box with an indication of the effort

spent. The result and effort will depend on the actor's performance, determined by motivation and ability.

5.8. Social Environment Model

The social environment model takes into account both the socio-organisational structure and the situation that influences the behaviour of the actors. The social environment model adopts an Organisational Behaviour (OB) model based on three levels: individual, group and organisation (Robbins 2003). The three levels in the OB model are building blocks, where each level is constructed on the previous level. Groups and teams grow on the foundations laid in the Individual level and Organisations grow on the foundations of Groups. The original refereed paper published by this research proposing this Social Environment Model is included in Appendix D.

The SIAABM includes the concept of teams, groups and organisations. Teams are made up of individuals that work very closely together and share the same goals. A Team Leader actor leads the team. Groups represent departments in the organisation. A group is led by a Manager actor and is made of one or more teams that work together sharing the same goal of the group. The organisation is made of one or more groups and is led by a Senior Manager actor. Figure 8 shows the Social Environment model in the context of the ACTS theory model.

Each actor is associated with a team, a group and an organisation. Actors in the same team have a higher probability of interaction and collaboration. However the model allows for interaction between actors from different teams, groups and organisations. The social environment model does not impose any limitation on the number of organisations, and multiple organisations that can be instantiated during simulation.

The model implements the concept that an individual's knowledge drives organisational knowledge creation (Nonaka 1994; see Organisational Knowledge Creation in Section 2.4.7). Group knowledge is the aggregation of the combined knowledge of its members. Similarly, the knowledge of the organisation comprises the people's tacit knowledge plus explicit knowledge captured in the form of documents, made available to all members of the organisation for individual learning. At the individual level actors can learn through the process of internalisation (Nonaka 1994; see Section 2.4.7) by

transforming explicit knowledge into their own implicit knowledge. To do so, the actor has to be motivated to dedicate time to the process of self-learning.

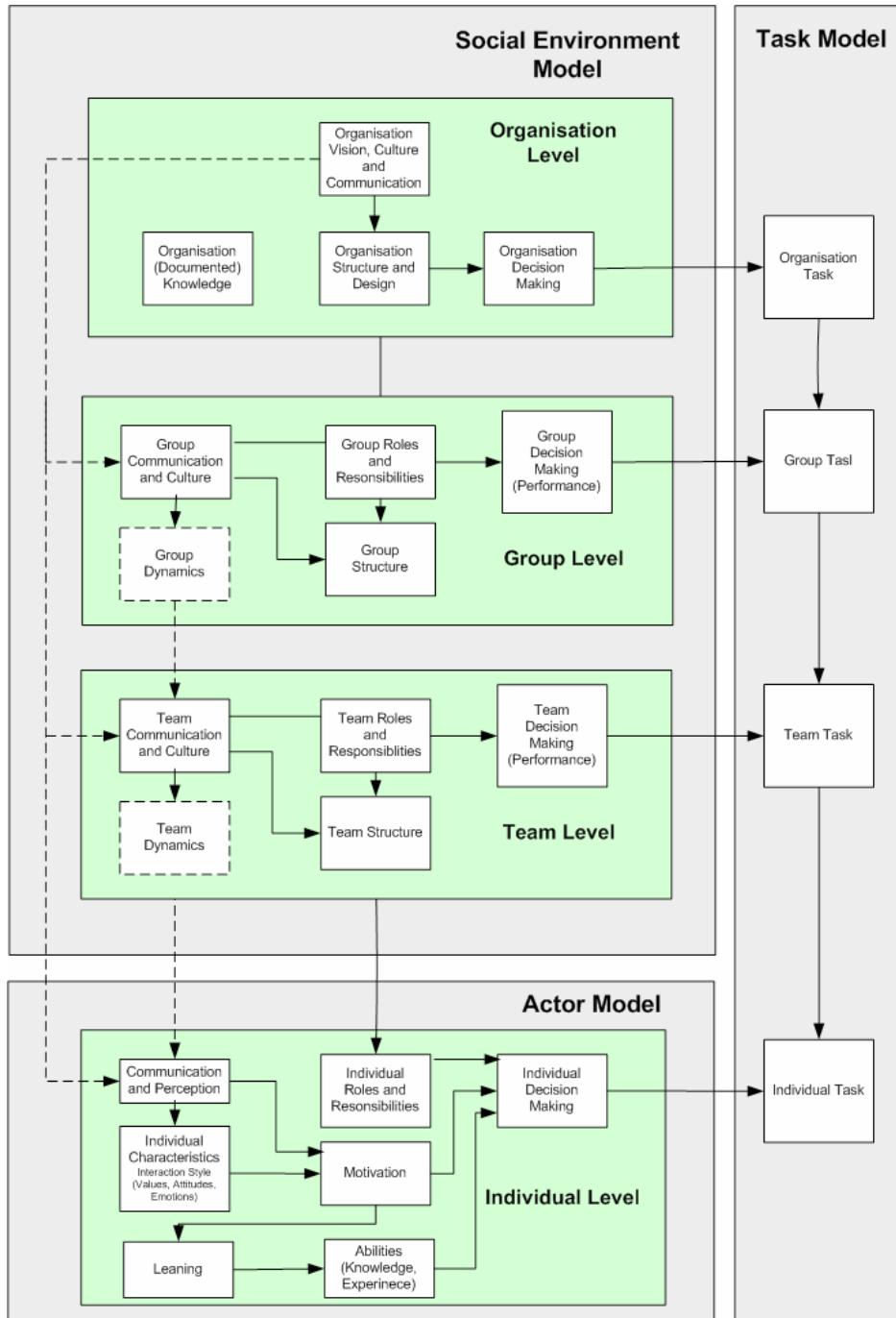


Figure 8 – Social Environment model

This research could not find in the literature any other agent-based model that represents organisational structures with the level of detail of the proposed model. The SIAABM provides mechanisms to create multiple organisations formed by individual actors with specific roles (Team Members, Team Leaders, Managers and Senior Managers) and

attributes (knowledge, experience, personality and motivation). The level of detail included in the SIABM provides a rich environment to model and simulate scenarios representing real software-intensive acquisitions.

5.8.1. Relationship Model

The relationship model, shown in Figure 9 expands BASP (Reynolds & Dixon 2000) to adapt to Axelrod's framework and the ACTS theory. Like BASP, the framework comprises *agents* and *connections*. Agents are then expanded to *actors* and *artefacts*; connections are expanded into *interactions*, *tasks* and *dependencies*.

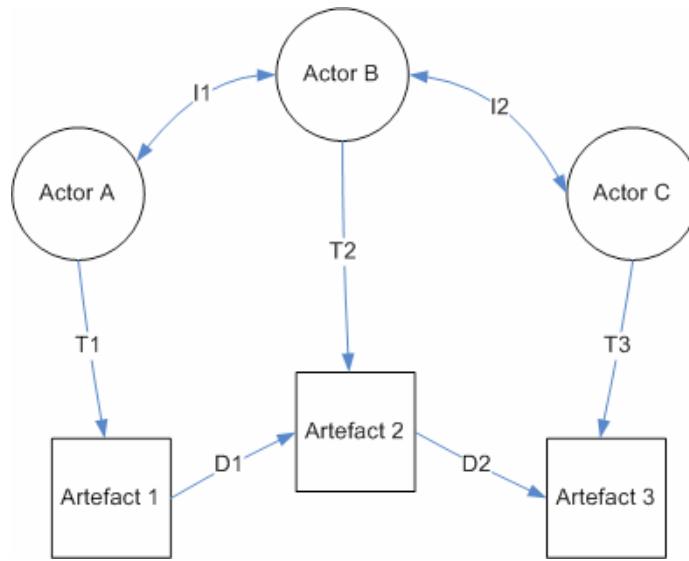


Figure 9 – Relationship Model

Actors and Artefacts

Actors are active agents and represent people. Actors have attributes that represent the actor's cognition through their knowledge, emotions and motivation. The set of attributes constitutes the actor's state and determine the actor's behaviour that results in the actor's actions. In the course of the simulation the actor's attributes may change, with a consequential change in behaviour.

Artefacts are passive agents and represent physical and conceptual objects that are manipulated, modified and created by actors. Artefacts, however, do not have cognition or behaviour. Artefacts are modelled with a set of attributes that form the artefact's state representing the artefact's ideal and actual values. The artefact's state changes as the actors perform their tasks. The difference between the ideal and actual value corresponds to the distortion or error present in the actual artefact.

Interactions, Dependencies and Tasks

Actors interact with other actors and with artefacts through specific connections. Connections are the way to form teams and organisations, assigning tasks to actors and establishing dependencies between artefacts. A connection between actors establishes an interaction; a connection between actors and artefacts defines a task; a connection between artefacts establishes a dependency.

Interaction models the process of socialisation and is the mechanism that allows the actors to expand their knowledge by sharing experience and knowledge with other actors (Nonaka 1994; see Organisational Knowledge Creation in Section 2.4.7). Interactions can occur formally, as part of the structure established by the socio-organisational design, or informally, through friendships and acquaintances. Either way, interactions influence the actor's behaviour to a certain degree, depending on the actors and the interaction itself. The interaction defines the relationship between the actors, being of authority (superior/subordinate), peer or acquaintance.

Dependencies establish a dependency factor between artefacts and are the basis to define the tasks that the actors will perform. For products, as a collection of artefacts, to be effective and satisfy the need, the dependency factors have to be maintained when the actors execute the tasks that are assigned to them.

Tasks define what the actors must do to create and modify artefacts. The framework proposes three types of tasks: transformation, discovery and rework. A transformation task creates artefacts given the input artefact and the transformation factor that represents the work the actor must perform. A discovery task is created and assigned to an actor for discovering distortions in artefacts already created. Rework tasks are in fact transformation tasks that modify artefacts to correct distortions discovered by discovery tasks. So that if a task is performed without distortions the actor must possess the nominal level of knowledge required. Tasks also have a nominal level of effort that must be applied by the actor. If less than the nominal effort is applied by the actor, the task will be performed with distortions.

An important aspect of connections is their activation, which determines when an interaction, task or dependency occurs. The activation of connections determines the sequence of events and is the mechanism that creates feedback loops in the social system.

Activation, Connections and Feedback

Connections have to be established and activated. A connection between an actor and an artefact is established when a task is created and it is activated when the task is assigned to an actor. Connections between actors are established when the interaction between the actors is created and it is activated when teams, organisations and acquaintances are established. Dependencies, the connection between artefacts, are always active; however, as artefacts do not have behaviours, the effect of dependencies will occur when actors execute tasks.

The activation of connections can cause feedback loops in the system that impact artefacts and actors. Figure 10 shows how feedback loops can occur. Discovery tasks create rework tasks that modify artefacts that through their dependencies impact other artefacts, causing more distortions and rework.

Formal discovery and rework tasks included in review and test activities do not influence the next product because the next product does not yet exist, and the rework occurs before the input to the next phase is base-lined. Informal discovery tasks occur after the product is base-lined, when undetected errors are discovered during the next project phase and are fed back to the previous phase causing rework of input and output artefacts in the current project phase. It is not uncommon to detect defects in artefacts produced in earlier phases and reworking these artefacts will impact artefacts that are connected to them through dependencies.

Feedback loops also occur on the social system and impact actors. The activation of formal and informal interactions between actors influences actors' behaviours. Cooperation is a positive form of interaction which can occur formally or informally. Cooperation increases shared knowledge and understanding, which in turn increases productivity and reduces errors and rework.

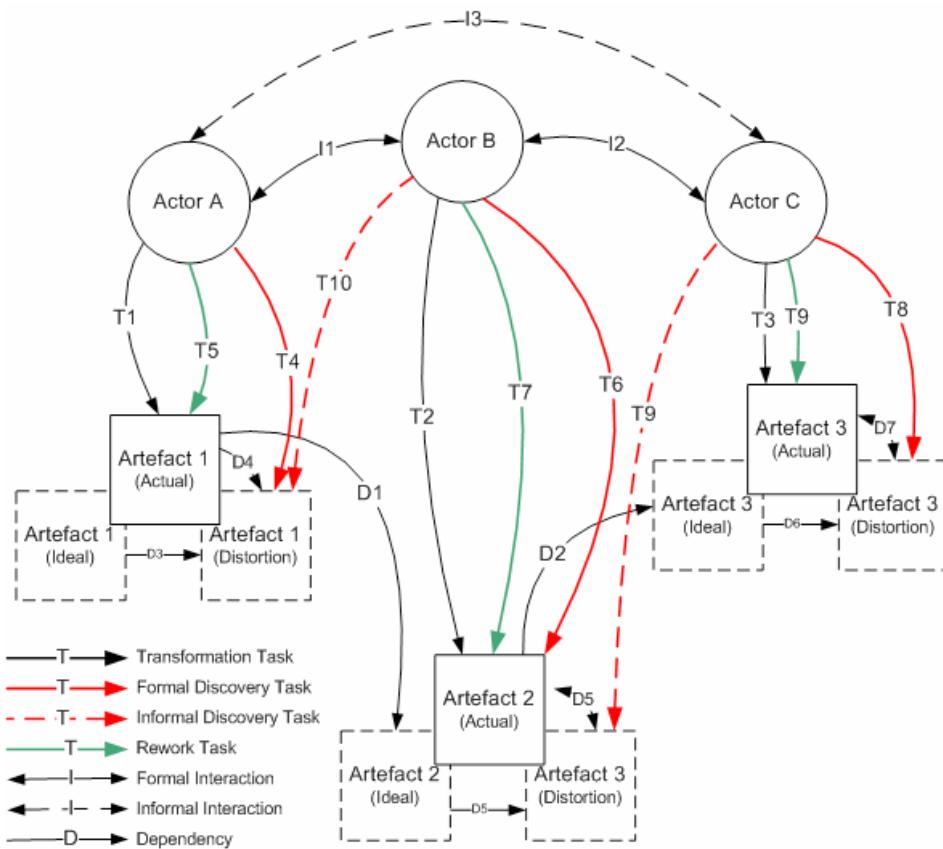


Figure 10 – Feedback Loops

The review process activates another form of interaction that causes feedback loops in the social system. When occurring in a constructive environment led by transformational leaders, reviews contribute positively to the project to achieve the desired goals. However, negative consequences would arise when the project leadership is predominantly transactional; that is, focused on close monitoring the performance of subordinates searching for errors and deviations (Thite 1999). As a consequence of task-oriented leadership, reviews would increase management pressure followed by behaviours that could reduce group morale and productivity, moving the project away from its objectives.

5.9. Modelling Variation

Variation will be modelled through the heterogeneity of the actors and the non-deterministic nature of their actions included in the cognitive agent model presented in Section 5.7. The actor's attributes and the probability of their actions are determined by configuration parameters at simulation. The actor's behavioural style determines the probability of its actions, including the ability to learn and forget. As the actor's

behaviour may change in accordance with its interactions, its actions may also change. The model also allows for actors randomly changing behaviour based on a probability of occurrence specified by a configuration parameter. All these features included in the model provide a rich source of an actor's variations that will drive variations in the observed parameters of cost, schedule, effectiveness and efficiency of the simulated software-intensive acquisition.

Actors can be specified individually or by a population defined by a probability of distribution. The actor's attributes that are subject to variation are behavioural style, knowledge, experience and motivation. Organisations are specified by the number of actors in each of the following roles: Senior Manager (one per organisation), Manager, Team Leader, Expert and Team Member. More on configuration parameters is presented in Chapter 6 and in Appendix A.

5.10. Summary

The chapter started with a review of modelling and the simulation of social systems, observing their strengths and limitations. The objectives of a model of software-intensive acquisitions were stated together with theories and hypotheses that support the model. The agent-based modelling approach was chosen as the most appropriate for the model objectives.

The Software-Intensive Acquisition Agent-Based Model (SIAABM) founded on the ACTS (Agent, Cognition, Task and Social environment) theory was proposed to meet the characteristics that are required to test the hypotheses about the system and variations in software-intensive acquisitions. Each of components of the model was presented, representing cognitive agents, the task and the social environment.

The SIAABM will be used in a computer-simulation environment to test the hypotheses about the software-intensive acquisition system and its variations formulated in Chapters 3 and 4. The simulation results are presented in Chapter 6.

Chapter 6: Exploration in a Simulated Environment

6.1. Overview

The Software-intensive Acquisition Agent Based Model Simulation (SIAABMSim) developed for this research is a *Repast 3.1*³⁷ Java application. SIAABMSim implements the Software-Intensive Acquisition Agent-Based Model (SIAABM) presented in Chapter 5. A description of SIAABMSim displays and configuration parameters is included in Appendix A.

6.1.1. A Brief Description of the Simulation Process

Simulation scenarios are specified by configuration files that define the task, the population of actors and how the actors are associated in teams, departments and organisations. What follows is a brief description of the simulation process implemented by SIAABMSim.

Actors, Teams, Departments and Organisations

Actors are created in accordance with a set of attributes that define the actor's role (Senior Manager, Manager, Team Leader, Team Member or Expert); behavioural style; knowledge; experience and motivation. Each organisation has one Senior Manager and one or more Managers, Team Leaders, Team Members and Experts. The number of Team Members is allocated evenly to Team Leaders and each Team Leader and their Team Members form one team. Similarly the number of Team Leaders is allocated evenly to Managers and Managers are allocated to the Senior Manager of the organisation. Each Manager and their teams form a department. The organisation has one or more departments and each department has one or more teams.

Actor's Behavioural Style

The actor's behavioural style attribute is used to implement the emotional model described in Section 5.7.4. The behavioural style is a number between 0.0 and 1.0 that represents the three main behavioural styles (Aggressive = 0.0; Constructive = 0.33; and Passive = 0.66) and three intermediate behavioural styles (Doer = 0.165; Nice = 0.495;

³⁷ Recursive Porous Agent Simulation Toolkit (Repast) is a Java based toolkit for agent-based modelling and simulation available from http://repast.sourceforge.net/repast_3.

and Neutral = 0.895). The actor's behavioural style determines the actor's actions in accordance with the probabilities specified by the simulation configuration. As the actor interacts with other actors during the simulation its behavioural style may change.

The actor's behavioural style determines the actor's actions in accordance with the probabilities specified by the simulation configuration. As the actor interacts with other actors during the simulation its behavioural style may change. The rates that determine how behaviour changes during the simulation are controlled by configuration parameters as a mechanism to calibrate the simulation.

Actor's Knowledge, Experience and Motivation

The actor's knowledge and experience attributes are used to implement the cognition model described in Section 5.7.5 and the motivation attribute adds to the behavioural model presented in Section 5.7.7. These attributes are also expressed as numeric values between 0.0 and 1.0.

Knowledge and experience determine the actor's ability to execute a task. If the actor has less than the required knowledge and experience the task will be executed with errors and the output artefact will contain distortions. The motivation attribute influences the actor's willingness to do the work, learn and collaborate. Lower motivation causes the actor to take longer to do the task. As the actor works, learns and interacts with other actors during the simulation its knowledge, experience and motivation may change. The actor may also forget what has been learnt. The rates of learning, forgetting and other attributes that can change during the simulation are controlled by configuration parameters as a mechanism to calibrate the simulation.

Interactions

Actors interact with other actors through several forms of interactions including Management, Leadership, Consultancy, Peer-to-Peer and Remote. Each type of interaction has its own attributes that define the nature and formality of the interaction. Management and Leadership are professional interactions between actors that have a boss-subordinate relationship; Consultancy is the interaction between an Expert actor and other actors; Peer-to-Peer is an interaction between Team Members; and Remote interactions are informal between actors of different teams, departments and organisations. The organisational structure specified by configuration files defines

professional and formal interactions. Remote and informal interactions are created by a probability of occurrence specified by configuration parameters.

Interactions are channels that the actors use to assign tasks, report progress, collaborate, praise and reprimand.

Transformations, Products, Tasks and Artefacts

All simulation scenarios use the same task instantiation, described in the next section. The task comprises six transformations, each with one input, and one output product. Each product has a number of artefacts. The output product of one transformation is the input product to the next. The responsibility to develop an output product is allocated to an organisation as a contract, comprising one input product, a transformation and one output product. The Senior Manager receives the contract and allocates artefacts to Managers and these distribute the artefacts to Team Leaders that will assign tasks to develop, review and rework output artefacts to Team Members. When all output artefacts are complete the output product is *baselined*³⁸ and passed to the next contract as the input product.

Transformations are two-dimensional matrices and each component of the matrix is a task. Tasks are assigned to Team Members that will apply their knowledge and experience to perform the task. The task is performed without errors when the actor's knowledge and experience matches what is required for the task and the actor applies the effort needed to complete the task.

The transformation matrices define the dependencies between products and artefacts. A task that modifies an output artefact can be executed only when all its dependencies, i.e. input artefacts, are complete and baselined.

Engineering Process and Development Life Cycle

The simulation implements a Systems Engineering process defined by the sequence of transformations. The development process comprises three tasks: development, review and rework. The effectiveness of each task depends on the ability of the actor that performs it. If the actor has less than the knowledge and experience to perform the task

³⁸ In a real project when a product is placed under configuration control it is said to be 'baselined', meaning that it is complete and ready for use. When a product is baselined in the simulation it is an indication that it can be used and the input for the next transformation.

the artefact will contain errors. The review task aims to find errors in the artefact. The errors found will be proportional to the ability of the actor that performs the review. When errors are found by the review task a rework task is assigned and again the effectiveness of the rework will depend on the knowledge and experience of the actor that performs the rework. Development and rework tasks add to the effectiveness of the artefact and are productive work. Reviews are support tasks that do not contribute directly to the effectiveness of the artefact and therefore are non-productive work. The effort spent on development and rework tasks increase efficiency while review tasks reduce efficiency.

All simulation scenarios for this thesis use the same task configuration, described further in this chapter. The development life cycle can be *sequential* or *evolutionary* and in one or many increments. When simulating a sequential life cycle, the completion of one development phase triggers the start of the next phase, e.g. at the end of the system specification, the system design starts, while the evolutionary life cycle will execute the complete development cycle, from need definition to the development of the software-intensive solution, to start the next complete development cycle. In the evolutionary life cycle the effectiveness of the final product, i.e. the software-intensive solution, is evaluated and helps to improve the next evolutionary increment by revisiting the need definition that flows to the next evolutionary phase.

The development phase may also detect errors in the input products and feedback the errors to the previous phase that reworks its output artefacts and issues a new baseline that becomes more work for the next phase. The more errors are informed to the previous phase the more work is required in the development phases that follow, creating a cumulative ‘snow-ball’ effect. The number of errors fed-back to the previous phase is controlled by a configuration parameter. Review and rework increase the effectiveness of the solution and also increase the effort and time to develop it.

Learning and Collaboration

If the actor does not have sufficient knowledge to complete the task a decision may be made to spend time learning or asking for help from another actor. The probability of an actor spending time learning, receiving or giving help is a function of the actor’s behavioural style.

As an actor self-learns or learns by collaborating with another actor they become better prepared to perform the task and the output artefact will contain less distortions. The actor will also be better prepared to perform reviews and rework and detect errors in input artefacts. As the population of actors learn, collective knowledge and experience increases and the acquisition system is better prepared to understand the need and the solution.

Time spent in learning and collaborating is accounted as non-productive work and reduces the efficiency of the acquisition.

Configuration Parameters

The simulation is configured by three files (see Appendix A for details): Organisation File defines actors, teams and organisations; Probabilities File determines the probability of an actor's actions; and, SimSettings File contains parameters that influence the whole simulation, including the development life cycle model, rates and limits.

The task configuration requires several files defining products artefacts and transformations. The task configuration is kept constant for all simulation scenarios tested for this research.

Simulation Results

The simulation state is shown dynamically as the simulation progresses and the final results are displayed on the screen or recorded in a file when the simulation finishes.

The simulation state is shown dynamically as the simulation progresses in three main windows: the *Environment Window*, the *Behaviour Window* and the *Performance Window*. The *Environment Window* shows the dynamic state of *actors*, *artefacts*, *tasks* and *interactions*. The *Behaviour Window* shows the collective dynamic behaviour of the *actors* over time which includes the average of *motivation* and the average of the three components (aggressive, constructive and passive) of the behavioural style of the *actors*. The *Performance Window* shows the progress of the task and the knowledge gap between what the tasks require and what is available in the population of actors. See Appendix A for details.

The final results are displayed on the screen or recorded in a file when the simulation finishes. The final results of the simulation include the effectiveness of the solution; the effort, duration and the efficiency of the development process. The simulation can be executed in multi-run mode without dynamic display and in this mode the results are recorded in a file for analysis. This method was used to collect the data shown graphically as the result of the simulation scenarios reported in this chapter.

6.2. Task Configuration

The task configuration for all simulation scenarios in this thesis is based on the specification and development of the Super Seasprite SH-2G(A) Helicopter that was being acquired by the Commonwealth of Australia for the Royal Australian Navy (RAN) until the acquisition was cancelled in 2008. The original refereed paper produced by this research proposing the task model presented in Section 5.6, and its application to the Super Seasprite SH-2G(A) Helicopter acquisition is included in Appendix B.

The first two levels of decomposition, comprising the roles and capabilities of the aircraft, are based on public information (DMO 2006; NOC 2006; Scott, Holdanowicz & Bostock 2004). The relative level of importance of roles and capabilities are hypothetical.

The Need

The Royal Australian Navy identified the need for a helicopter to support operations from the ANZAC class frigates. Two primary roles for the aircraft were identified:

- a. Increase the ship's effectiveness by expanding Surveillance (SV) capability;
- b. Contribute to the ship's combat capabilities providing Anti-Ship Warfare (ASH) and Anti-Submarine Warfare (ASB).

The secondary roles for the aircraft comprised Search-and-Rescue (SR), Support Missions (SM) such as medical evacuation and Crew Training (CT). The need also determines that the aircraft is to be operated by a crew of two, day and night and under adverse weather conditions.

The hypothetical relative contribution of the aircraft roles, shown in column P1 in Table 21, was estimated using the Analytic Hierarchy Process (AHP) (Saaty 2000), introduced

in Section 5.6.1. The 6x6 Priority Matrix is a pair comparison between each of the six roles performed by the aircraft, and P_1 is the normalised principal eigenvector of the Priority Matrix. The diagonal of the Priority Matrix corresponds to the comparison of each of the six roles with itself and therefore is one. The other components of the Priority Matrix are the comparison of the role in the column with the role in the row of the matrix. All the numeric calculations were performed with *Mathematica* 7 (software tool by Wolfram Research, Student Edition).

Table 21 – Aircraft Roles

Aircraft Roles	P_1	SV	ASH	ASB	SR	SM	CT
Surveillance (SV)	0.41	1	3	3	7	9	9
Anti-Ship Warfare (ASH)	0.28	1/3	1	3	7	9	9
Anti-Submarine Warfare (ASB)	0.18	1/3	1/3	1	5	7	9
Search and Rescue (SR)	0.07	1/7	1/7	1/5	1	3	7
Support Missions (SM)	0.04	1/9	1/9	1/7	1/3	1	5
Crew Training (CT)	0.02	1/9	1/9	1/9	1/7	1/5	1

The need (Table 22) defines six roles and twelve capabilities: HMI for crew of two (HMI); Aircraft Monitor & Supervision (MS); Mission Preparation Support (MPS); Mission Debrief Support (MDS); Electronic Navigation (NAV); Radio Communications (COM); Surveillance Sensors (SVS); Electronic Warfare (EW); Tactical Data Management (TDM); Tactical Navigation (TNM); Anti-Ship Weapons (WAS); and Anti-Sub Weapons (WASb). Table 22 shows the need in the form of the dependency between roles and capabilities.

Table 22 – The Need Transformation Matrix (T1)

Roles		Capabilities												
		P_2	HMI	AMS	MPS	MDS	NAV	COM	SVS	EW	TDM	TNM	WAS	WASb
SV	0.41		0.1	0.08	0.08	0.02	0.1	0.12	0.2	0.1	0.1	0.1	0	0
ASH	0.28		0.1	0.08	0.08	0.02	0.1	0.12	0.1	0.1	0.1	0	0.2	0
ASB	0.18		0.1	0.08	0.08	0.02	0.1	0.12	0.1	0.1	0.1	0	0	0.2
SR	0.07		0.1	0.08	0.08	0.02	0.1	0.12	0.1	0	0	0.4	0	0
SM	0.04		0.1	0.08	0.08	0.02	0.1	0.12	0.1	0	0	0.4	0	0
CT	0.02		0.1	0.08	0.08	0.02	0.1	0.12	0.1	0.08	0.08	0.08	0.08	0.08

The shaded column P1 represents the hypothetical contribution of the aircraft's roles, as shown in Table 22. The set of non-shaded numbers defines a 6x12 transformation matrix T1 and represents the contribution of each capability to the aircraft's roles. The sum of the numbers in each row is 1.0 representing the whole capability. Taking Surveillance (SV) capability as an example, Surveillance Sensors (SVS) is the most important capability with a factor of 0.2, followed by Communications with a factor of 0.12. Other capabilities also contribute to Surveillance with lesser importance and Surveillance does not depend on Anti-Ship Weapons (WAS) or Anti-Sub Weapons (WASb) capabilities and their weight is consequently zero. The other rows corresponding to the contribution of the 12 capabilities to the other aircraft roles were estimated in the same way. Although the numbers in T1 were estimated without AHP, AHP could have been used to improve accuracy (each row requires a 12x12 Priority Matrix).

The shaded row P2 represents the hypothetical contribution of each capability to the need as a whole, calculated by equation (6.1) below, where P1 and P2 are respectively the input and output product vectors and T1 is the transformation matrix:

$$(6.1) \quad \mathbf{P2} = \mathbf{P1} \cdot \mathbf{T1}$$

The elements of the transformation matrix are tasks whose execution is affected by the ratio of the agent's useful knowledge and the knowledge required to execute it. It is interesting to observe that there are multiple dependencies between roles and capabilities. If Surveillance Sensors (SVS) is partially or not available, for example, all roles defined for the aircraft will be affected. Also, to express the need requires knowledge in multiple areas. To specify the need for Surveillance Sensors (SVS) requires not only knowledge of surveillance sensors but also of all of the aircraft's roles. Lack of knowledge would cause deficiencies in expressing the need.

The expressed need is presented in the form of an Operational Concept or similar document, represented by equation (6.1).

The Problem

The expressed need is the input to the next level of decomposition. The problem is expressed in the form of what is required to resolve the expressed need. Capabilities endure further analysis to reveal products required to resolve the problem. The analysis

continues until all products and their relationship are completely defined and specified. There are 26 primary products identified for the SH2G(A) helicopter, comprising HMI devices, Main Data Processor, Mission Data Loader/Recorder, Navigation Devices, Communication Equipment, Aircraft Sensors, Radar, Forward Looking Infra Red (FLIR), Electronic Warfare (EW), Threat Warning, Counter Measure Dispenser (CMDS), Link-11 and various Weapons.

Capabilities are expanded into primary products and their contribution to the whole is estimated. Being P2 and P3 respectively the Capabilities vector input and the Primary Products vector output, the transformation matrix T2 is a 12x26 matrix such that:

$$(6.2) \quad \mathbf{P3} = \mathbf{C2} \cdot \mathbf{T2}$$

The problem analysis is completed when the solution is defined and proposed in the form of a Function Performance Specification or similar document, represented by equation (6.2).

The Solution

The analysis of the problem and the proposed solution are the input to the next transformations. The solution comprises a series of transformations that produce enabling and operational products. Enabling products are the specification and design for the system, hardware and software, while operational products are hardware, software and procedures that will be delivered with the final solution.

Each of the 26 primary products identified in the proposed solution requires a system specification and design, as well as specification and design for human and hardware interfaces. The output of specification and design is then 156 enabling products. The specification and design process is represented by equation (6.3), where P3 is the Product input vector, P4 is the Specification and Design output vector and T4 is the 26x156 transformation matrix.

$$(6.3) \quad \mathbf{P4} = \mathbf{P3} \cdot \mathbf{T4}$$

The system under development calls for the integration of products that require the development of dedicated software. As the focus of this paper is on software-intensive systems, the transformations that follow, address only software products.

Each of the 26 primary products depends on software and requires software specification and design. The output of software specification and design is then 52 enabling products. The specification and design process is represented by equation (6.4), where P4 is the System Specification and Design input vector, P5 is the Software Specification and Design output vector and T5 is the 156x52 transformation matrix.

$$(6.4) \quad P5 = P4 \cdot T5$$

The specification and design of software products should contain all information required to enable the production of software components. Each of the 26 primary products requires the development of dedicated software products. The output of the software productions is then 26 operational products. The production of software products is represented by equation (6.5), where P5 is the Software Specification and Design input vector, P6 is the Software products output vector and T6 is the 52x26 transformation matrix.

$$(6.5) \quad P6 = P5 \cdot T6$$

System specification and design documents intend to bring knowledge and information from previous domains into the software domain. When this intent is not achieved, lack of information and knowledge will propagate into the software domain and will carry distortions and omissions from specifications created in previous domains. To detect and correct these anomalies requires knowledge beyond the software domain, which usually is no longer or easily available.

Five Development Phases

The task configuration represents five engineering phases of the software-intensive acquisition:

- a. Operational Concept Definition, transforms the statement of *need* into a set of *capabilities* required to satisfy the *need*;
- b. Functional Performance Specification, transforms *capabilities* into a set of *system functions* required to implement the *capabilities*;
- c. System Analysis and Design, transforms *system functions* into a *system specification and design*;

- d. Software Analysis and Design, transforms the *system specification and design* into the *software specification and design*.
- e. Software Development, transforms *software specification and design* into *software components*.

Effectiveness and Efficiency

The effectiveness of the solution is given by the sum of the contribution of the end products, which in the ideal case is 1.0. Equation (6.6) represents the effectiveness of the solution, where in this example p_i represents the 26 components of the vector P6:

$$(6.6) \text{ Effectiveness} = \sum_{i=1}^{i=26} p_i$$

Unless distortions are reduced, whether at the end of the development cycle in the form of testing and rework, or as an integral part of the engineering process, 100% effectiveness cannot be attained.

Equation (6.7) represents the efficiency of the system as the ratio between the actual and ideal effort, where actual effort includes the effort spent to produce enabling end products plus the effort spent on additional enabling tasks and rework:

$$(6.7) \text{ Efficiency} = \frac{\sum_{n=1}^{n=6} PEffort(P_n)}{\sum_{n=1}^{n=6} AEffort(P_n)}$$

Chinese Whispers Effect

The application of the task model to the Super Seasprite SH-2G(A) Helicopter shows that to produce 26 software operational products requires the development of 252 enabling products in the form of operational concept, functional performance and other specification and design documents. Test products do not impact the solution directly and were not included in the example.

Each element of the transformation matrices, if not zero, is a task that requires knowledge of input and output products, spread over 300 areas of knowledge. Equations (6.1) to (6.5) represent five transformations required to produce software products. In

summary, to produce 26 outputs needed some 250 intermediate outputs in a five stage process. The distortion level of each transformation will determine how correct the end product will be if compared with the ideal solution. This reflects the *Chinese Whispers* effect caused by the propagation of distortions and omissions due to the lack of useful knowledge. Distortions will also occur due to the poor quality of communication and if less than the nominal effort is applied. Wrong estimation can be associated with lack of knowledge and skills and contribute to reduce the effectiveness of the system.

Distortions and rework caused by lack of useful knowledge, whether in engineering or management, could partially explain schedule delays and cost overrun in software-intensive projects.

6.3. Promising Results

Earlier versions of SIAABMSim shown promising results published in two original refereed papers included in Appendices C and D for reference. Both simulations used the task configuration described in Section 6.2. The model of cognitive agents although simpler than the current state of SIAABMSim includes the essential features of adaptation and the actors can learn and change behaviour as they perform tasks and interact with other actors.

6.3.1. Influence of Knowledge and Interaction Styles

The first application of SIAABMSim was intended to assess the impact of knowledge and interaction styles in the time required to complete the acquisition, from the definition of the need to the implementation of software products (see Appendix C for reference).

Simulation Scenario

The transformations are complete when the effectiveness of the output product reaches its nominal value of 1.0. This condition makes the project continue to the end regardless of cost and schedule. Terminating the project when a pre-determined cost or duration is reached could simulate the failure of the project. If the effectiveness of the final product did not achieve a useful level, the project would be seen as a failure.

A team of actors lead by a Team Leader actor performs each transformation. There are five teams, each executing a transformation T_i . The number of actors was arbitrarily defined and is kept constant for each team.

The point of reference is the performance of actors with ‘super experience’ and passive interaction style. The time taken to execute the tasks to completion, when all teams are formed by Super-Passive agents, is assigned the normalised value of 1.0.

Each test scenario executes the same tasks defined by the transformation matrices T_i . The number of actors in each team is not changed, but the experience and interaction style is varied. The combination of experience and interaction styles gives 36 scenarios. There are homogeneous scenarios, where all the agents in the team have the same type of attributes (varied within a uniform distribution), such as Senior-Aggressive, Junior-Constructive or Super-Passive; and heterogeneous scenarios, where the agents are created with two types of attributes with the same proportions, such as Super-Senior/Aggressive-Constructive or Senior-Junior/Constructive-Passive.

Simulation Results

Figure 11 shows the normalised duration for the 36 scenarios. When all actors possess super knowledge, learning and cooperation is not required to execute the task; their work is always productive and they are not influenced by the behaviour of other agents or their own. Super actors represent the ideal case with the lowest duration and effort (since the number of agents is constant) required to execute the task.

The results in Figure 11 show an example of non-linear effect on the duration of tasks caused by available experience, knowledge and interaction styles of people executing the tasks, which makes accurate estimation and planning extremely difficult. Constructive teams perform better than teams with aggressive or passive behaviour, and teams with half of the required experience would take three times longer to execute the task.

Homogeneous teams of junior actors have the lowest performance, requiring more time to execute the task due to the fact that they have to spend effort learning. Interaction styles have no major influence in the team performance because in a homogeneous team of inexperienced actors, there are no other actors with more experience and knowledge who could help the less experienced junior actors.

Teams with senior actors perform better and are more influenced by interaction styles. Senior actors have a lower gap between their knowledge and what is required, a difference that can be acquired through self-learning and collaboration with other senior actors. Heterogeneous teams with senior, junior and super (expert) actors are a closest representation of real projects, and the simulation demonstrates that interaction styles can influence the performance of teams and the outcomes of the acquisition as whole.

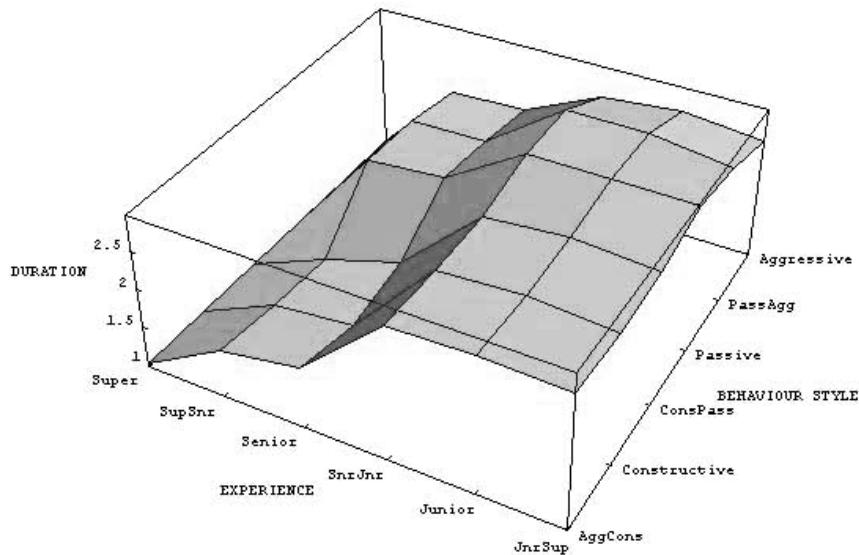


Figure 11 – Duration, Experience and Interaction Styles

Simulation results show that the interaction style of Team Leaders impact the performance of the team. Aggressive Team Leaders are likely to make their teams become passive, while constructive team leaders are likely to influence their teams to become constructive or to be even more constructive, leading to better performance. Passive team leaders will transmit their behaviour to the team. These results are supported by the dynamic of group interaction in accordance with individual and group interaction styles (Cooke & Lafferty in Cooke & Szumal 1994) (see Life Style Inventory (LSI) and Group Style Inventory (GSI) introduced in Section 2.4).

The results from simulation suggest that teams with less experience take longer to perform the task and under constructive leadership the team can be motivated to learn, collaborate and complete the task sooner. However, it is interesting to observe that often in practice aggressive management is put in place as a strategy to bring the project in alignment with the plan and the result is the opposite. Aggressive management causes

passive reaction from the team, reduces the conditions for learning and collaboration and the situation becomes worse.

6.3.2. Influence of Learning through Collaboration

The second application of SIAABMSim intended to assess the impact of learning through collaboration is the effectiveness of the solution and the time required to complete the acquisition, from the definition of the need to the implementation of software products (see Appendix F for reference).

Simulation Scenario

The simulated scenarios represent the ideal and a more realistic case. The simulation starts with the definition of the ideal artefacts, dependencies and tasks. Actors have their own knowledge, motivation, interaction style, role and responsibilities. The ideal actor possesses perfect knowledge and executes the assigned tasks without distortions. A more realistic actor has less than ideal knowledge and introduces distortions in the tasks that are executed. The actor's interaction style determines the willingness to learn and cooperate with other actors. Team performance and interaction with other actors influence the actor's motivation and productivity.

The ideal scenario takes the representation of the ideal artefacts and their dependencies and creates ideal actors capable of performing the tasks without distortions and within the allocated time. The effort, duration and effectiveness of the ideal case are normalised and represented by 1.0. The results from the ideal scenario are used as the reference to assess the results obtained from the realistic scenario.

The actors in the realistic scenario have less knowledge and experience to execute the task. The realistic scenario is used to execute the simulation in two states: first without collaboration, and then allowing the actors to collaborate.

Simulation Results

The ideal scenario produces the ideal product effectiveness at ideal duration since there is no need for discovery and rework tasks. The realistic scenario without collaboration requires more rework to achieve the required effectiveness. The realistic scenario that allowed the actors to collaborate achieved the required effectiveness in less time.

Through collaboration individual knowledge is shared and becomes collective knowledge, which is an emergent property of social systems. Every time a task is executed the actors learn and the errors produced are reduced. Learning and cooperation decrease the number of iterations, which reduces time and effort to complete the tasks and the project.

Although the simulation results are not evidence that learning and collaboration are better approaches than individual learning, they show that agent-modelling and simulation can be used to investigate and explore the behaviour of social systems.

6.4. Testing Hypotheses

The hypotheses formulated about the software-intensive acquisition system and its variations were tested in a computer-simulated environment using the SIAABMSim. The following are simulation conditions and the results.

6.4.1. Ideal Scenario

The ideal scenario is used as a reference to compare the performance of realistic scenarios (see Section 2.3.1). The ideal scenario comprises actors that have perfect knowledge and experience and are motivated to do the task. The actors will work ‘like robots’ without being influenced by other actors and maintaining constant behaviour and productivity.

The objective of the test is to identify what is the ideal number of actors and the number of increments in a *sequential development*³⁹ life cycle to develop the software-intensive solution in the shortest time. The number of ideal actors is expected to be determined by the dependencies in the development tasks. The ideal number of actors should provide the best plan to execute the tasks concurrently resulting in the shortest development time.

The simulation will test two organisational configurations. All ideal Team Member actors will be able to perform any task in the process and the single organisation will be responsible for all products, artefacts and transformations in the process. The second configuration will be with three organisations: one responsible for expressing the need

³⁹ Sequencial development mimics the waterfall life cycle (see Engineering Process and Development Life Cycle in Section 6.1.1).

and capability development; the second organisation will take the capabilities and transform them into a system specification and design; the third organisation will receive the system specification and design and develop the software specification, designing and the software components that should satisfy the need.

The configuration with three organisations represents a typical case in software-intensive acquisitions. The first organisation corresponds to the user and the customer; the second organisation is the contractor that performs the systems engineering activities; and the third organisation represents a sub-contractor that develops the software.

Simulation Scenario for Testing Number of Actors

The simulation scenario uses the ideal configuration parameters (see Appendix A for details): ideal simulation settings (SimSettings-Ideal); ideal probabilities (Probabilities-Ideal) and ideal organisations (Organisations-1Org-Ideal and Organisations-3Orgs-Ideal).

The development life cycle is sequential with one increment, i.e. typical ‘big-bang waterfall’. The number of Team Member actors varying from 100 to 1000 in ten configurations is tested for one and three organisations. Team Member actors represent the engineers and developers that specify and implement the software-intensive solution. The number of actors for other roles (Team Leaders and Managers) is proportional to the number of Team Member Actors (one Team Leader to five Team Members and one Manager for three to four teams). Each configuration is executed in multiple runs, 50 times, and the mean value of duration is recorded.

After the ideal number of actors is found the test is repeated with the ideal number of actors for one and three organisations and this time varying the number of incremental developments from one to ten. Again, each configuration is executed in multiple runs, 50 times, and the mean value of duration and effort are recorded.

Simulation Results

The simulation results for schedule performance as a function of the number of actors for one and three organisations are shown in Figure 12. The results are normalised to the best schedule performance. The best number of Team Member actors to achieve the shortest schedule is in the order of 300 for one organisation and 750 for three

organisations. The result also shows that within limits, a less aggressive schedule will be achieved with less people.

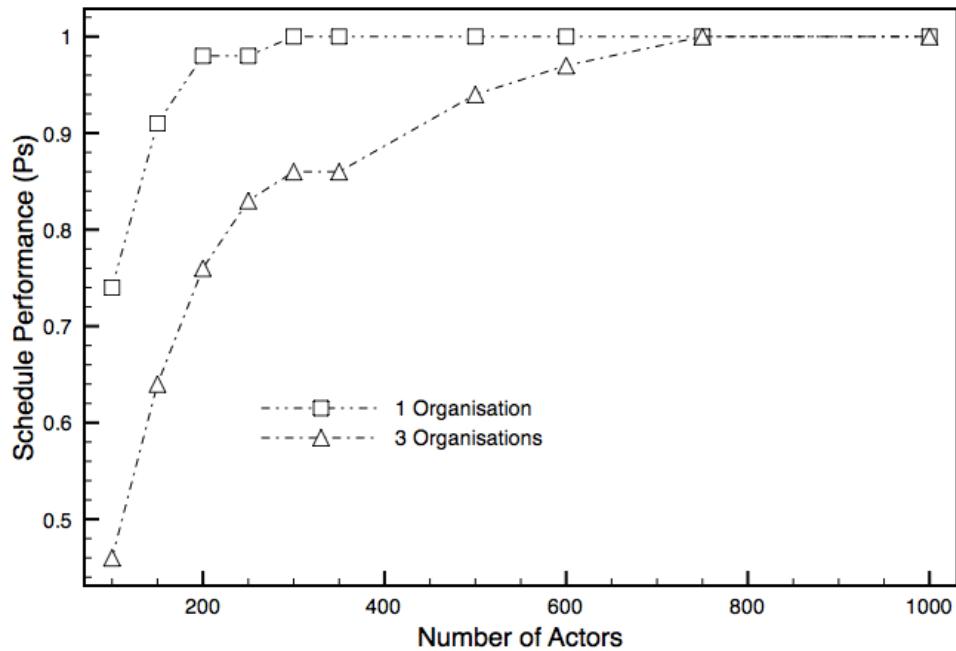


Figure 12 – Schedule Performance

Simulation Scenario for Testing Number of Increments

The test of schedule and cost performance as a function of the number of development increments was executed with 300 actors in one single organisation and 750 actors in three organisations.

Simulation Results

The results are shown in Figure 13. The best schedule performance (Ps) is achieved by three organisations, 750 Team Member actors and six development increments, and it is used as the reference ($Ps = 1.0$). The best Ps produces the best concurrent execution of tasks. The simulation assumes that actors that are waiting for a task assignment or for a task dependency to be complete do not impose cost to the project. The cost performance (Pc) is similar for both configurations because the volume of work is determined by the task, which is the same for the two configurations of organisations of ideal actors. As the number of development increments increases Pc decreases because of the greater spread of overheads created by management and team leadership.

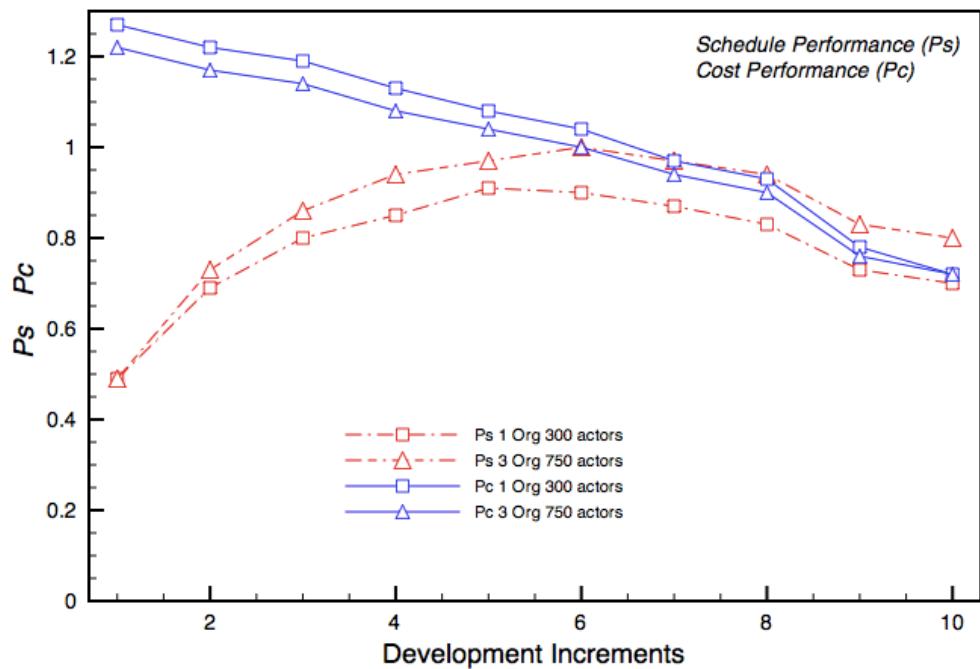


Figure 13 – Schedule and Cost Performance

Figure 14 shows the effectiveness achieved by three organisations with 750 actors performing the development in one single increment, i.e. a waterfall life cycle. Figure 15 shows the effectiveness achieved by the same organisation developing the solution in six sequential increments. The incremental approach starts delivering increments of the software product much earlier than the waterfall approach. In both cases there is no knowledge gap and the progress of the development is close to linear because the system is ideal and there is no need for learning and rework. The efficiency of the process is determined by the volume of non-productive tasks, i.e. reviews and management. The final product effectiveness is the same for the waterfall and incremental development because the actors are ideal. The project duration is shorter if the development is in six increments because tasks can be executed in parallel.

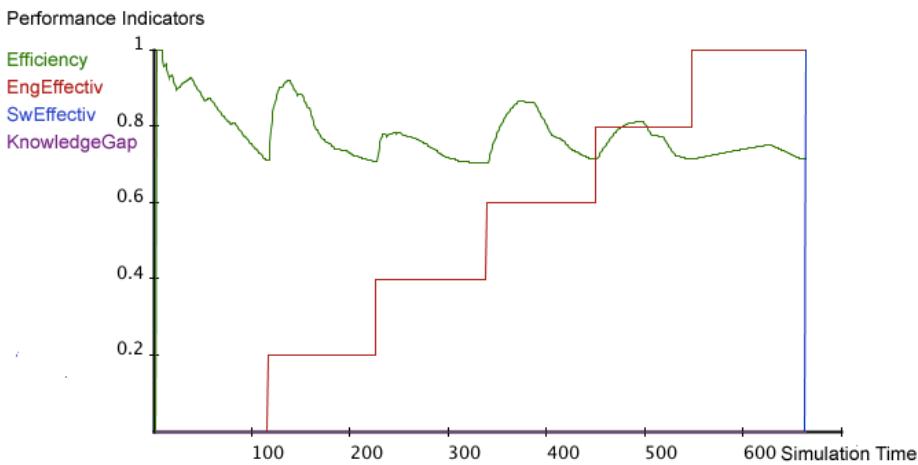


Figure 14 – Effectiveness 3 Organisations Waterfall Development

The five steps in red shown in Figure 14 correspond to the five development phases that produce intermediate products that will lead to the development of the final software product represented by the blue vertical line.

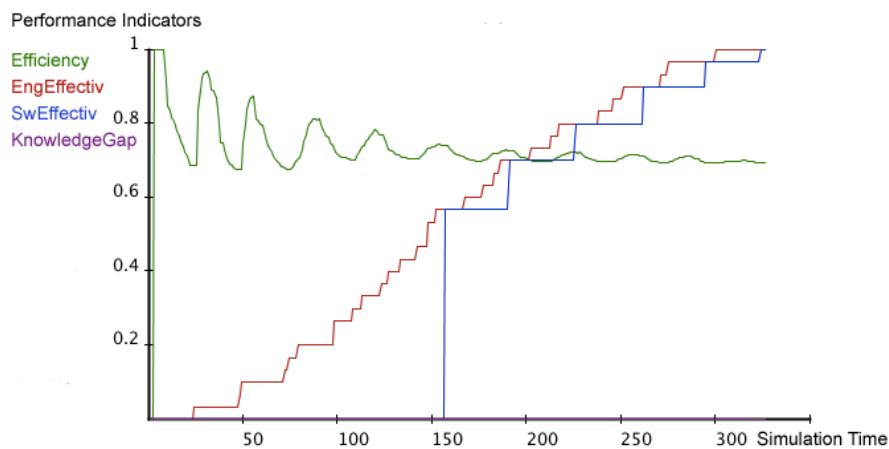


Figure 15 – Effectiveness 3 Organisations Incremental Development

The six steps in blue shown in Figure 15 correspond to the six incremental deliveries of the software product.

The configuration with three organisations and 750 Team Member actors and a sequential life cycle with six development increments represents a realistic case and will be used to test the first hypothesis.

6.4.2. First Hypothesis

The objective in testing the first hypothesis is to investigate the impact of the absence of essential knowledge in software-intensive acquisitions.

Simulation Scenario

The simulation performed adopts a sequential development life cycle with six incremental steps, and with three organisations. The number of actors is 953 distributed as 750 Team Members, 150 Team Leaders, 50 Managers and 3 Senior Managers (also named CEO for historic reasons). Team Members are the engineers that perform the tasks. Team Leaders, Managers and Senior Managers allocate the tasks and monitor progress. The test executes 40 scenarios with the same configuration of organisations varying the knowledge, experience and behavioural type of the actors. The configuration specifies the mean value for the attributes with a standard deviation of 0.05. Each test is executed 50 times in batch multi-run mode.

Figure 16 shows one of the configuration files for the test: behaviour = 0.17 (Doer); knowledge = 0.7; experience = 0.7 and motivation = 1.0. The standard deviation is 0.05 for all attributes (see Organisations File Format⁴⁰ in the Appendix A).

ORG	ORG1	0.2	Nil	Application					
CEO	1	0.17	0.05	0.3	0.05	0.3	0.05	1.0	0.05
Manager	3	0.17	0.05	0.3	0.05	0.3	0.05	1.0	0.05
Expert	0	0.33	0.0	1.0	0.0	1.0	0.0	1.0	0.0
TeamLeader	9	0.17	0.05	0.3	0.05	0.3	0.05	1.0	0.05
TeamMember	45	0.17	0.05	0.3	0.05	0.3	0.05	1.0	0.05
ORG	ORG2	0.2	Capabilities	SysEng	SysArchitect				
CEO	1	0.17	0.05	0.3	0.05	0.3	0.05	1.0	0.05
Manager	20	0.17	0.05	0.3	0.05	0.3	0.05	1.0	0.05
Expert	0	0.33	0.0	1.0	0.0	1.0	0.0	1.0	0.0
TeamLeader	60	0.17	0.05	0.3	0.05	0.3	0.05	1.0	0.05
TeamMember	300	0.17	0.05	0.3	0.05	0.3	0.05	1.0	0.05
ORG	ORG3	0.2	SoftEng	SoftDev					
CEO	1	0.17	0.05	0.3	0.05	0.3	0.05	1.0	0.05
Manager	27	0.17	0.05	0.3	0.05	0.3	0.05	1.0	0.05
Expert	0	0.33	0.0	1.0	0.0	1.0	0.0	1.0	0.0
TeamLeader	81	0.17	0.05	0.3	0.05	0.3	0.05	1.0	0.05
TeamMember	405	0.17	0.05	0.3	0.05	0.3	0.05	1.0	0.05

Figure 16 – Organisations File - 3Orgs-B017-K03-E03-M10-SD005-750

⁴⁰ The rows starting with ORG specify the name of the organisation, the probability of interaction between actors, and the tasks assigned to that organisation. The rows that follow define the population of actors for that organisation in the form of Role, Number of actors in that role followed by the mean value and standard deviation of Behaviour, Knowledge, Experience and Motivation for that population.

The simulation is driven by the way that the actors behave in accordance with probability of action defined by configuration parameters (see Section 5.7.4). Figure 17 shows the Probabilities File used for the simulation (see Appendix A for format details). The table shows that the probability of a constructive actor offering help is 80% while for an aggressive actor is 2%. The probabilities in the table are defined by assumptions about how actors with a particular behavioural style are likely to behave. The simulation results are deeply influenced by the Probabilities File.

For this test Manager and Senior Manager actors keep their initial behavioural style throughout the simulation, while Team Leader and Team Member actors may change behaviour in response to interactions with other actors.

Header	Aggressive	Doer	Constructive	Nice	Passive	Neutral
Start	0.05	0.10	0.20	0.10	0.05	0.01
Respond	0.50	0.80	0.90	0.80	0.70	0.50
AskHelp	0.05	0.20	0.50	0.30	0.10	0.05
OfferHelp	0.02	0.05	0.80	0.10	0.05	0.01
AcceptHelp	0.30	0.40	0.80	0.50	0.20	0.10
Help	0.05	0.10	0.50	0.30	0.10	0.05
Praise	0.01	0.05	0.30	0.20	0.05	0.0
Reprimand	0.50	0.30	0.05	0.01	0.0	0.0
Learning	0.10	0.20	0.30	0.10	0.05	0.01
Forgetting	0.05	0.10	0.15	0.20	0.25	0.30
Motivated	0.10	0.15	0.20	0.15	0.10	0.15
Demotivated	0.20	0.25	0.40	0.25	0.10	0.15
Working	0.90	0.98	0.95	0.90	0.80	0.85
Overtime	0.70	0.50	0.30	0.10	0.05	0.10
Aggressive	0.50	0.40	0.30	0.20	0.1	0.0
Constructive	0.10	0.20	0.80	0.30	0.10	0.0
Passive	0.03	0.05	0.10	0.15	0.25	0.20
BAggressive	0.0	0.0	0.0	0.0	0.0	0.0
BConstructive	0.0	0.0	0.0	0.0	0.0	0.0
BPassive	0.0	0.0	0.0	0.0	0.0	0.0
MngAggressive	0.0	0.0	0.0	0.0	0.0	0.0
MngConstructive	0.0	0.0	0.0	0.0	0.0	0.0
MngPassive	0.0	0.0	0.0	0.0	0.0	0.0
TLAggressive	0.50	0.40	0.30	0.20	0.1	0.0
TLConstructive	0.20	0.20	0.5	0.30	0.10	0.0
TLPassive	0.03	0.05	0.10	0.15	0.25	0.20
QualityMotiv	0.2	0.3	0.7	0.6	0.55	0.5
CostSchedMotiv	0.8	0.7	0.3	0.4	0.35	0.5
ExplicitReward	0.7	0.6	0.3	0.4	0.55	0.5
ImplicitReward	0.3	0.4	0.7	0.6	0.35	0.5

Figure 17 – Probabilities File

There are five groups of eight scenarios each. Each group corresponds to one of five behavioural styles: Aggressive, Doer, Constructive, Nice and Passive. Neutral style was not included in the test.

Each group is configured with eight levels of knowledge and experience: 1.0; 0.9; 0.8; 0.7; 0.6; 0.5; 0.4 and 0.3, corresponding to a knowledge gap from 0.0 to 0.7.

Simulation Results

The normalised results⁴¹ relative to the ideal scenario are shown in Figure 18 to Figure 21. It is important to observe that all the actors have similar knowledge and experience and the best way to improve knowledge and experience is through self-effort. Actors can still collaborate, but the gap of knowledge between the actors brings little benefit.

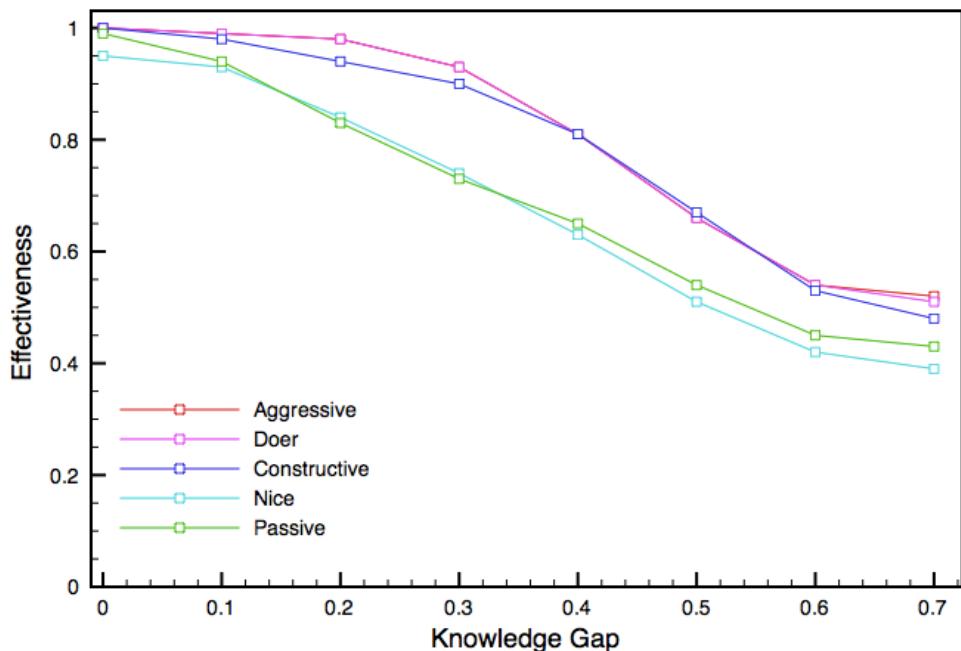


Figure 18 – Effectiveness x Knowledge Gap and Behavioural Style

The effectiveness result in Figure 18 shows that actors classified as Aggressive and Doer perform better than all other behavioural styles. The reason is that although all actors have similar experience and there is a limited gain in overall knowledge, Aggressive and Doer actors concentrate on having the job done, while Constructive actors spend time interacting with other actors and Passive and Nice actors waste time

⁴¹ The normalised results reflect the average of 50 runs for each scenario. The results of each run, the average and standard deviation for each of the 40 scenarios are included in the Appendix G (DVD).

rather than doing productive work. These are the fundamental characteristics of actors in each behavioural style.

If the gap of knowledge is less than 0.2 Aggressive and Doer actors manage to perform as well as ideal actors by applying effort to learn what is needed to do the work. Constructive actors collaborate even when they could fill the knowledge gap by self-learning and the time spent interacting unnecessarily with other actors is not applied to do the work, hence the lower performance. Nice and Passive actors are likely to waste time and are not interested in acquiring new knowledge, reflected in the lowest performance.

When the knowledge gap is greater than 0.2 the effectiveness starts to drop for all behavioural styles. The reason is that there is not enough time to acquire the required knowledge by self-learning and collaboration with more knowledgeable and experienced actors would be necessary.

The results in Figure 19 and Figure 20 are a confirmation that lack of knowledge extends schedule and increases costs, as both schedule performance (Ps) and cost performance (Pc) decrease when the knowledge gap increases. The additional time and effort is spent in learning what is needed to do the task.

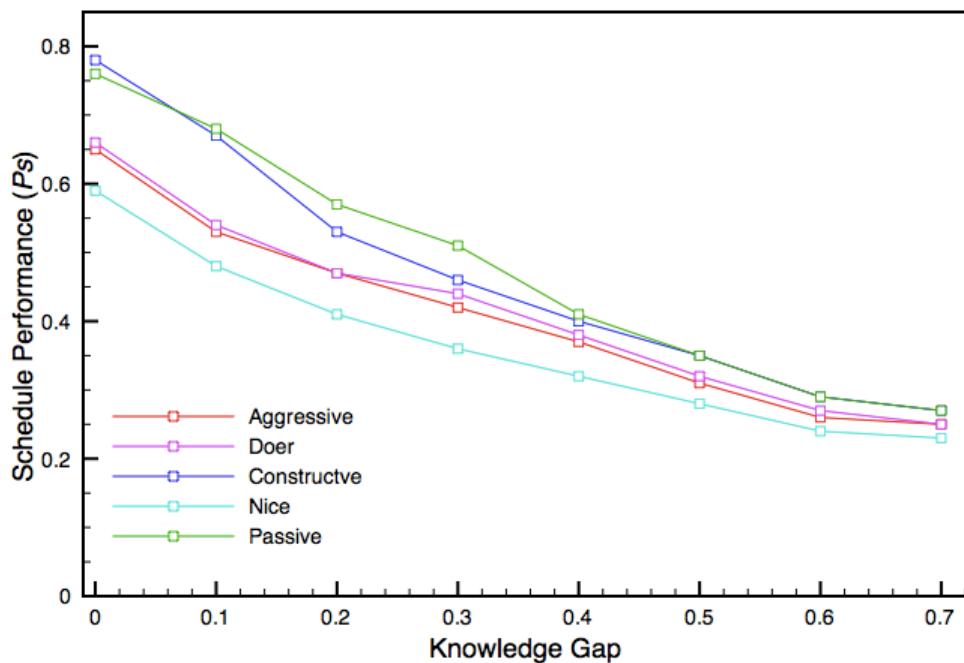


Figure 19 – Schedule Performance (Ps) x Knowledge Gap and Behavioural Style

The schedule performance and cost performance graphics show that passive actors perform well in cost and schedule and better than all the other behavioural styles in cost performance. The high performance is caused because passive actors do not apply the required effort to do the task, resulting in the lowest effectiveness among all behavioural styles (see Figure 18). The case of passive actors reflects the illusion of increasing schedule and cost performance by not executing the work properly, a not unusual phenomenon in real projects.

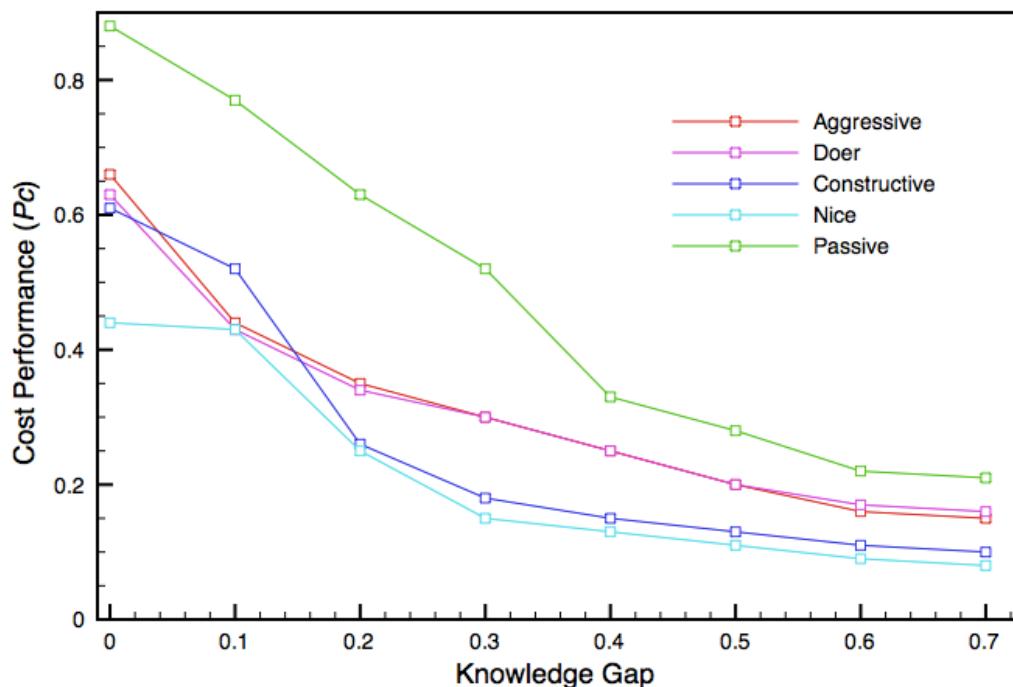


Figure 20 – Cost Performance (P_c) x Knowledge Gap and Behavioural Style

The efficiency results in Figure 21 show that the efficiency achieved by Aggressive, Doers and Passive actors reflects the intrinsic efficiency of the process determined by organisational structure and engineering processes. These actors demonstrate a higher probability of working on the assigned task and not as much spending time interacting with other actors or learning. Besides of high effectiveness, passive actors produce low product effectiveness as result of lack of knowledge and their low probability of seeking help from other actors and spending time acquiring the required knowledge. The efficiency in the order of 0.65% is determined by the ratio of effort spent on productive

tasks and the total effort. Non-productive tasks are those of management, leadership and engineering reviews (see Engineering Process and Development Life Cycle in Section 6.1.1).

The efficiency shown by Constructive and Nice actors is further reduced by the effort spent on interactions or simply wasted. The fluctuations in efficiency can be justified by the dynamics of a complex socio-technical system, although they may not be easily and categorically explained. The fluctuations shown in the graphic reflect the difficulty, seen in practice, to accurately plan and predict the behaviour of software-intensive acquisitions, and in particular when a considerable gap of knowledge exists.

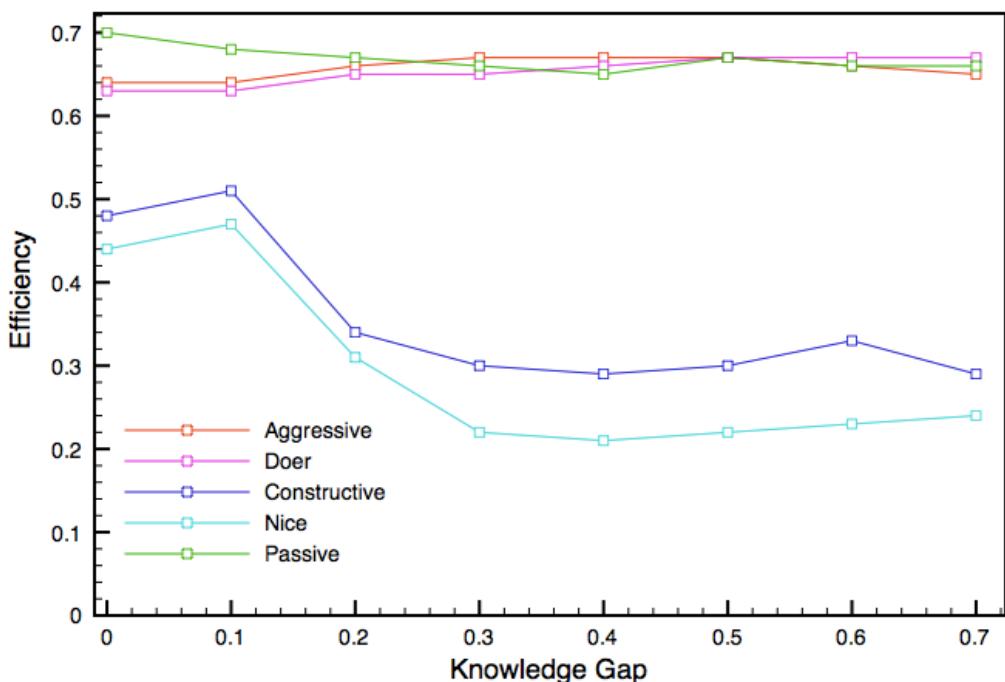


Figure 21 – Efficiency x Knowledge Gap and Behavioural Style

In conclusion, the simulation results confirm the first hypothesis, i.e., lack of knowledge required to engineer the software-intensive solution increases cost, extends schedule and reduces the effectiveness of the solution. The results also showed that cost, schedule, effectiveness and efficiency are not only influenced by the available knowledge and experience but also by the behavioural style of the actors, which confirms the impact of group interaction styles on problem-solving groups (Cooke & Szumal 1994), discussed in Section 2.4.6, and the results of the development of on-board software for the Boeing 777 (DAF 2000 Appendix Q), as discussed in Section 3.6.1.

6.4.3. Second Hypothesis

Testing the second hypothesis will investigate the impact of systemic structures that facilitate learning behaviours on the cost, schedule and effectiveness of software-intensive acquisitions.

Simulation Scenario

The simulation scenario is based on the scenario used to test the first hypothesis (three organisations, 750 Team Member actors and six increments of sequential development). The scenario is modified to create a collaborative environment with the addition of Expert actors and constructive management, as a form of systemic structure that facilitates learning. Expert actors possess ideal knowledge and experience and present a Constructive behavioural style. A total of 150 Expert actors, one allocated for each team, are added to the scenario. Managers maintain a Constructive behavioural style throughout the simulation. The test executes 40 scenarios structured in the same way as for the first hypothesis, and each test is executed 50 times in batch multi-run mode.

Simulation Results

The normalised results⁴² of the simulation are shown in Figure 22 to Figure 25. The results from the previous simulation are shown with dotted lines for comparison.

Figure 22 shows that except for Passive actors all the other populations presented an increase of the effectiveness of the solution, resulting from the collaboration between Experts and Team Member actors. Passive actors maintain the same effectiveness whether or not the environment fosters constructive and learning behaviours, because Passive actors have a lower probability of interacting, asking for and accepting help.

The results from the simulation are intuitive and can be explained by the impact of group interaction styles on problem-solving groups (Cooke & Szumal 1994), discussed in Section 2.4.6. The positive effects of collaborative environments have been documented in case studies reporting successful projects as seen on the development of on-board software for the Boeing 777 (DAF 2000 Appendix Q), discussed in Section 3.6.1. However, reports of unsuccessful projects lack an explanation of the root causes of failure and in particular how gaps of knowledge and less than constructive

⁴² The normalised results reflect the average of 50 runs for each scenario. The results of each run, the average and standard deviation for each of the 40 scenarios are included in the Appendix G (DVD).

behaviours may have influenced the project outcome. The simulation results suggest that knowledge and behaviour influence project success and failure, and should be taken into consideration during planning and adequately managed in the course of the project.

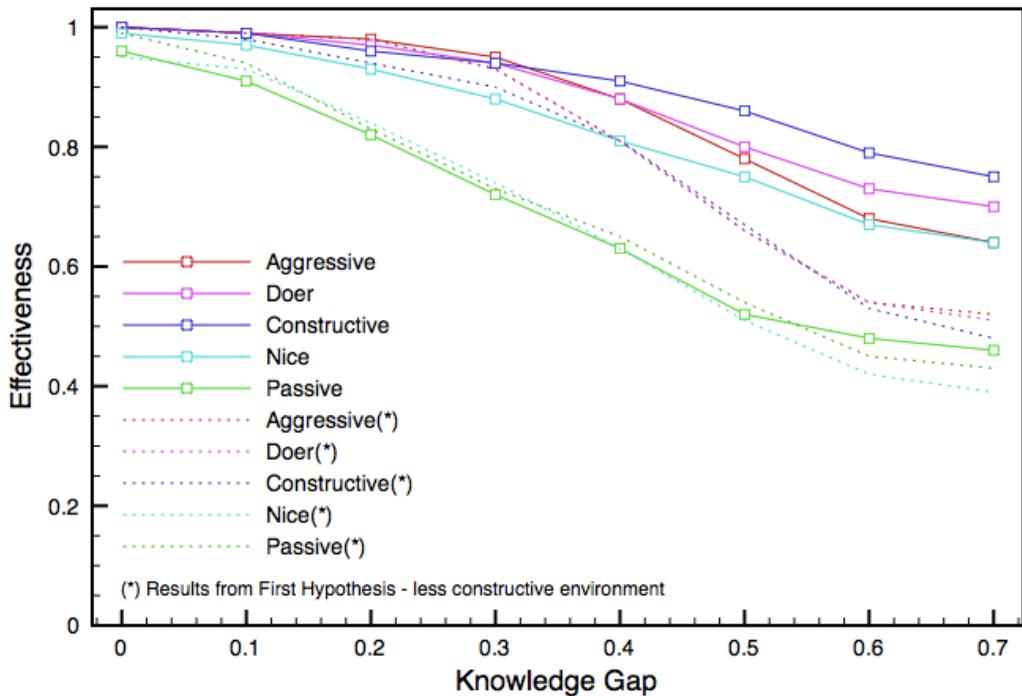


Figure 22 – Effectiveness x Knowledge Gap and Behavioural Style

A similar result is shown for schedule performance (P_s) in Figure 23, although passive actors also present an increase of schedule performance if the knowledge gap is less than 0.2. Although numeric values from the simulation should not be taken as quantitative truth, the results suggest that there is a level of knowledge gap beyond which schedule delays will reach unacceptable levels and become worse if the population of actors demonstrate passive behaviour. Overall, collaboration with more knowledgeable and experienced actors is more effective than self-learning, reducing the time required to acquire essential knowledge to perform the task, which is expected when the experts are collaborative.

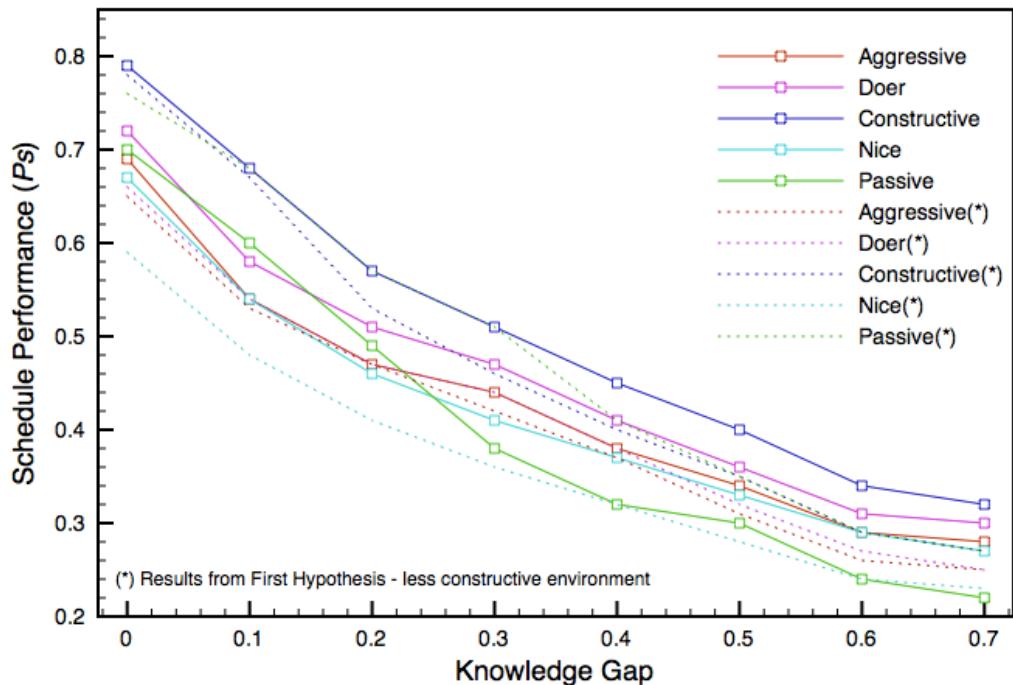


Figure 23 – Schedule Performance (P_s) x Knowledge Gap and Behavioural Style

The results of cost performance (P_c) shown in Figure 24 vary for each population. Aggressive and Doer actors show a decrease in performance if the knowledge gap is less than 0.3 and improve performance if the knowledge gap is larger. Again, being aware of the limitations of quantitative results, the simulation suggests that the behavioural style of the population and the knowledge gap influence cost performance, and there is a threshold knowledge gap below which the influence of behaviour is more accentuated.

The results can be explained by the fact that these actors prefer to work alone when they feel they can do the task and don't like to collaborate with other actors, unless in situations when they are unable to perform the task without help. Passive and Nice actors continue showing the lowest performance for the reasons already discussed.

The population of Constructive actors showed a consistent increase in cost performance that is even better when the knowledge gap is 0.4 or greater. The result demonstrates that even by adding 150 Expert actors into the system there was an overall reduction in cost. Collaboration is therefore more effective and cost attractive than self-learning.

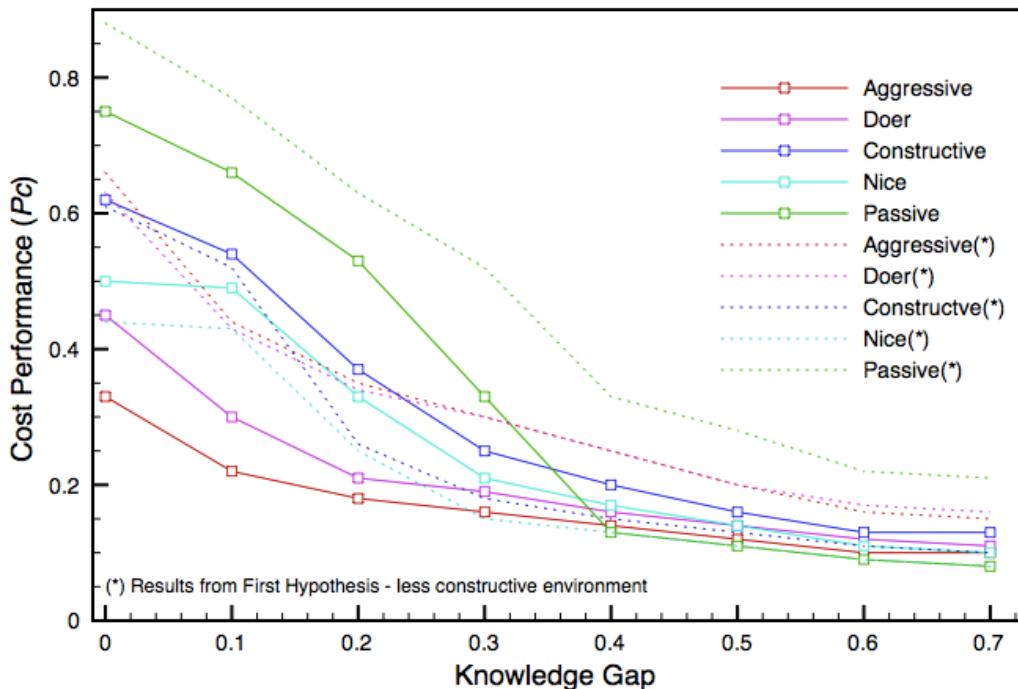


Figure 24 – Cost Performance (P_c) x Knowledge Gap and Behavioural Style

The efficiency results shown in Figure 25 also vary for each population of actors. The high efficiency of Passive actors when the knowledge gap is less than 0.3 is not meaningful since the corresponding effectiveness is low. The high efficiency of passive actors is caused by their low probability of learning and interacting with other actors, as discussed in the first hypothesis. As expected, the population of Passive and Doer actors show a consistent efficiency decrease for the same reasons that caused a decrease of cost performance. Constructive and nice actors are more efficient in a collaborative environment for low knowledge gaps, although with a reduction in cost performance. For higher gaps of knowledge, efficiency is lower because more effort is expended in collaboration.

The results from the simulation suggest that behaviour and knowledge have a higher impact on efficiency in a constructive environment. The fluctuations in efficiency are greater than what was observed from testing the first hypothesis (see Figure 21) and can be justified by even greater dynamic variations of a more constructive environment that promotes higher levels of interactions between actors. As already discussed in the simulation results for the first hypothesis, fluctuations in efficiency may be justified but cannot be definitively explained.

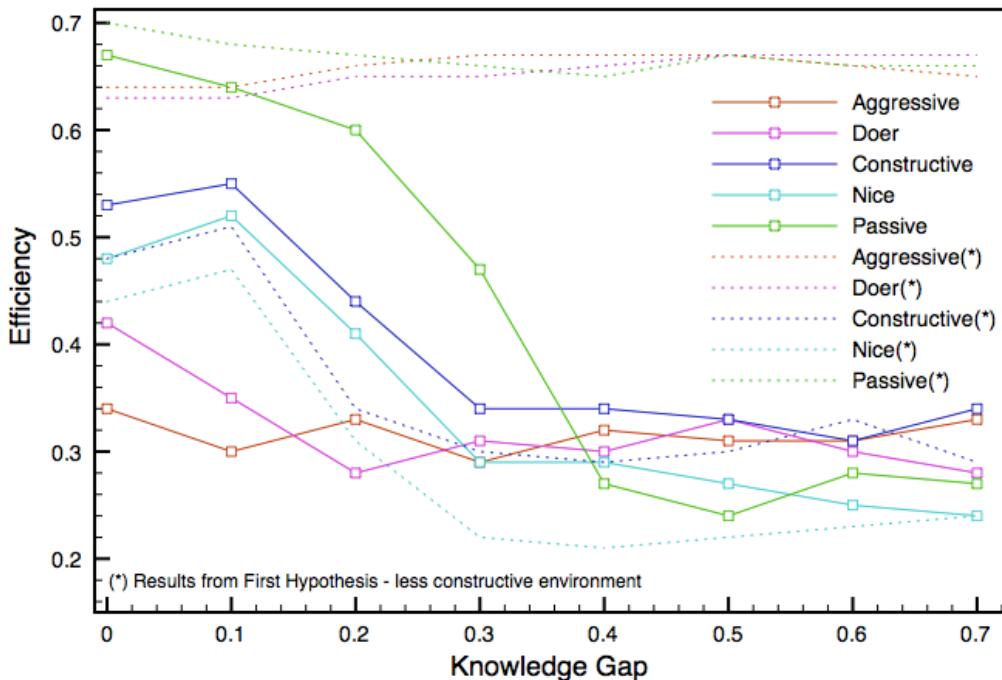


Figure 25 – Efficiency x Knowledge Gap and Behavioural Style

Although constructive environments create conditions for greater effectiveness when there is a high gap of knowledge, higher fluctuations in efficiency create even greater challenges for management. As efficiency is associated with less waste and therefore higher profit, management strategies that are focused on profit in projects displaying these conditions, will often make short-term decisions that may jeopardise the overall objectives of software-intensive acquisitions.

It is important to observe that the observed parameters of effectiveness, efficiency, cost and schedule respond non-linearly to the attributes of knowledge, experience and behaviour present in the population of actors, as effectiveness, efficiency and cost and schedule performance drop more rapidly as the knowledge gap increases.

The simulation scenario represents one of many possible systemic structures that foster learning behaviours, and in this case the simulation results confirm the Second Hypothesis, i.e., systemic structures that facilitate learning behaviours improve cost and schedule performance and the effectiveness of the solution of software-intensive acquisitions.

6.4.4. Third Hypothesis

The simulation scenarios and results from testing the first and second hypotheses are sufficient for testing the third hypothesis.

The tests kept the task constant and varied the population of actors representing a wide variation of knowledge, experience, and behavioural style. The results from testing the first and second hypotheses showed that variations in the population of actors cause a wide variation of effectiveness, efficiency, cost and schedule.

In conclusion the simulation results confirm the third hypothesis, i.e., variations in people's knowledge, experience, personality and motivation cause variations in the effectiveness, cost and schedule of software-intensive acquisitions. Higher levels of knowledge and experience and constructive behaviours available in the population of actors impact positively on effectiveness, cost and schedule performance. The conclusion reflects the reality of knowledge-based work where there is no machine or industrial process involved. The knowledge-based processes are imbued in people and therefore subject to the variation in people.

6.4.5. Fourth Hypothesis

Testing the fourth hypothesis investigates the impact of systemic structures in the reduction of people's variations and in observed variations of cost, schedule and effectiveness of the solution in software-intensive acquisitions.

Simulation Scenario

The simulation scenario contains three organisations with a total of 750 Team Member actors that was also used to test the first and second hypotheses. The scenario was modified to increase the variation of actors in the population by increasing the standard deviation of the attributes of the Team Member actors in the population, shown in Figure 26 (see the Organisations File Format in the Appendix A).

Organisations-3Orgs-750-Fourth-Hypothesis-Non-Collaborative.txt									
ORG	ORG1	0.2	Nil	Application					
CEO	1	0.0	0.05	0.5	0.2	0.5	0.2	0.7	0.2
Manager	3	0.0	0.05	0.5	0.2	0.5	0.2	0.7	0.2
Expert	0	0.33	0.0	1.0	0.05	1.0	0.05	1.0	0.0
TeamLeader	9	0.0	0.05	0.5	0.2	0.5	0.2	0.7	0.2
TeamMember	45	0.5	0.2	0.5	0.2	0.5	0.2	0.7	0.2
ORG	ORG2	0.2	Capabilities	SysEng	SysArchitect				
CEO	1	0.0	0.05	0.5	0.2	0.5	0.2	0.7	0.2
Manager	20	0.0	0.05	0.5	0.2	0.5	0.2	0.7	0.2
Expert	0	0.33	0.0	1.0	0.05	1.0	0.05	1.0	0.0
TeamLeader	60	0.0	0.05	0.5	0.2	0.5	0.2	0.7	0.2
TeamMember	300	0.5	0.2	0.5	0.2	0.5	0.2	0.7	0.2
ORG	ORG3	0.2	SoftEng	SoftDev					
CEO	1	0.0	0.05	0.5	0.2	0.5	0.2	0.7	0.2
Manager	27	0.0	0.05	0.5	0.2	0.5	0.2	0.7	0.2
Expert	0	0.33	0.0	1.0	0.05	1.0	0.05	1.0	0.0
TeamLeader	81	0.0	0.05	0.5	0.2	0.5	0.2	0.7	0.2
TeamMember	405	0.5	0.2	0.5	0.2	0.5	0.2	0.7	0.2

Organisations-3Orgs-750-Fourth-Hypothesis-Collaborative.txt									
ORG	ORG1	0.2	Nil	Application					
CEO	1	0.33	0.05	0.5	0.2	0.5	0.2	0.7	0.2
Manager	3	0.33	0.05	0.5	0.2	0.5	0.2	0.7	0.2
Expert	9	0.33	0.0	1.0	0.05	1.0	0.05	1.0	0.05
TeamLeader	9	0.33	0.05	0.5	0.2	0.5	0.2	0.7	0.2
TeamMember	45	0.5	0.2	0.5	0.2	0.5	0.2	0.7	0.2
ORG	ORG2	0.2	Capabilities	SysEng	SysArchitect				
CEO	1	0.33	0.05	0.5	0.2	0.5	0.2	0.7	0.2
Manager	20	0.33	0.05	0.5	0.2	0.5	0.2	0.7	0.2
Expert	60	0.33	0.0	1.0	0.05	1.0	0.05	1.0	0.05
TeamLeader	60	0.33	0.05	0.5	0.2	0.5	0.2	0.7	0.2
TeamMember	300	0.5	0.2	0.5	0.2	0.5	0.2	0.7	0.2
ORG	ORG3	0.2	SoftEng	SoftDev					
CEO	1	0.33	0.05	0.5	0.2	0.5	0.2	0.7	0.2
Manager	27	0.33	0.05	0.5	0.2	0.5	0.2	0.7	0.2
Expert	81	0.33	0.0	1.0	0.05	1.0	0.05	1.0	0.05
TeamLeader	81	0.33	0.05	0.5	0.2	0.5	0.2	0.7	0.2
TeamMember	405	0.5	0.2	0.5	0.2	0.5	0.2	0.7	0.2

Figure 26 – Organisations’ File for testing the Fourth Hypothesis

The population of Team Member actors is created as a normal distribution with 0.5 as the mean value (M) for knowledge (K), experience (E) and behavioural style (B) and 0.2 as the standard deviation (S). In the non-collaborative environment all managers and leaders are aggressive and there are no experts. In the collaborative environment the population of Team Member actors remains the same, but all managers and leaders are constructive and one constructive Expert actor is added to each team. In this test Manager actors can change their behaviour in response to interactions with other actors and in accordance with a Probabilities file that reflects a more realistic situation (see the Probabilities-Real file included in the Appendix A for reference). The large number of

actors in the population (953 and 1103 actors respectively in the non-collaborative and collaborative environments) brings statistical significance to the test.

The test is executed as *evolutionary development*⁴³ with six increments with and without collaboration. The collaborative environment is a systemic structure that facilitates learning behaviours, where there are experts willing to help and constructive managers and leaders.

The simulation will be performed under three test conditions:

- a. The simulation is executed until maximum effectiveness is achieved without imposed constraints of cost and schedule.
- b. The simulation terminates when a target effectiveness (E) of 0.9 is achieved without imposed constraints of cost and schedule.
- c. The simulation terminates when the imposed constraint of schedule performance (Ps) of 0.18 is achieved, corresponding to 5.5 times the duration of the ideal acquisition (see the Ideal Scenario in Section 6.4.1).

Each test scenario (Non-Collaborative (a), (b) and (c); Collaborative (a), (b) and (c)) is executed 50 times, and each execution creates a new population of actors maintaining the same probability distribution of actor attributes.

The analysis of results compares changes in the mean value (M) and standard deviation (S) of the actor attributes, and draws conclusions about how these changes impact cost performance (Pc), schedule performance (Ps) and the effectiveness (E) of the solution.

Simulation Results

Table 23 shows the minimum, maximum, mean value and standard deviation for E , Pc and Ps for 50 executions of each test scenario.

The effectiveness achieved by the collaborative scenario in all three tests conditions are higher than the effectiveness achieved by the non-collaborative scenario. The variation in effectiveness is also lower for the collaborative scenario. The results can be explained by the fact that collaborative environments create systemic structures that promote

⁴³ Evolutionary development mimics the evolutionary life cycle (see Engineering Process and Development Life Cycle in Section 6.1.1).

learning and collaboration and raise the knowledge and experience of the population of actors and thus the effectiveness of the solution. The results in Table 23 also show that both collaborative and non-collaborative environments occasionally achieved the target effectiveness, although the average effectiveness achieved by the collaborative scenario is higher.

Table 23 – Observed Variations of Effectiveness, Cost and Schedule

Test Condition	Non-Collaborative Environment Scenario				
		Minimum	Maximum	Mean (M)	Std Dev (S)
(a) No Constraints	Effectiveness (E)	0.550	0.980	0.878	0.090
	Cost Performance (P_c)	0.051	0.104	0.070	0.012
	Schedule Performance (P_s)	0.099	0.149	0.121	0.012
	Collaborative Environment Scenario				
		Minimum	Maximum	Mean (M)	Std Dev (S)
	Effectiveness (E)	0.790	0.990	0.919	0.050
(b) Target Effectiveness E = 0.90	Cost Performance (P_c)	0.053	0.078	0.064	0.006
	Schedule Performance (P_s)	0.120	0.173	0.146	0.013
	Non-Collaborative Environment Scenario				
		Minimum	Maximum	Mean (M)	Std Dev (S)
	Effectiveness (E)	0.530	0.910	0.830	0.101
	Cost Performance (P_c)	0.052	0.192	0.089	0.037
(c) Schedule Constraint P _s =0.18	Schedule Performance (P_s)	0.103	0.203	0.141	0.026
	Collaborative Environment Scenario				
		Minimum	Maximum	Mean (M)	Std Dev (S)
	Effectiveness (E)	0.720	0.920	0.887	0.043
	Cost Performance (P_c)	0.055	0.097	0.073	0.010
	Schedule Performance (P_s)	0.127	0.245	0.179	0.028
Test Condition	Non-Collaborative Environment Scenario				
		Minimum	Maximum	Mean (M)	Std Dev (S)
(c) Schedule Constraint P _s =0.18	Effectiveness (E)	0.530	0.930	0.774	0.101
	Cost Performance (P_c)	0.082	0.215	0.128	0.030
	Schedule Performance (P_s)	0.182	0.182	0.182	0.00
	Collaborative Environment Scenario				
		Minimum	Maximum	Mean (M)	Std Dev (S)
	Effectiveness (E)	0.780	0.970	0.884	0.050
	Cost Performance (P_c)	0.066	0.080	0.072	0.003
	Schedule Performance (P_s)	0.182	0.182	0.182	0.00

In all three test conditions the non-collaborative scenario performed better with respect to cost than the collaborative scenario. It is important to observe that the collaborative scenario has 150 actors in addition to the non-collaborative scenario. These actors are the experts that are constructive and knowledgeable and are the source for raising the knowledge and experience of the team. As the performance of the team improves part of the cost of Expert actors is offset by the reduction of errors and the lessened need for reworking tasks.

The collaborative scenario presented better effectiveness of the solution in all test conditions and better schedule performance when there is no imposed schedule constraint. Expert actors coach other actors but do not directly work on the solution.

Therefore the improvement in schedule performance can be justified by the improvement in the quality of work performed by Team Members as result of collaboration with Expert actors.

The results show that there is an overlap of the effectiveness achieved by the two scenarios. Within limits the non-collaborative environment can be as effective as the collaborative, and the collaborative environment may show low effectiveness levels when there is no collaboration. The collaborative environment, however, has a higher probability of achieving higher effectiveness, demonstrated by a higher mean value (M) and lower standard deviation (S) for effectiveness.

The overlap in the level of effectiveness achieved by both scenarios can be explained by the variation present in the initial population of actors. The population of Team Members is created with large variations in behaviour, knowledge and experience in accordance with the mean value ($M=0.5$) and standard deviation ($S=0.2$) specified for these attributes. As a consequence, the population will contain a variety of actors that range from constructive and knowledgeable, to passive and less knowledgeable and anything in between. Constructive and knowledgeable actors show a high probability of transferring their knowledge and experience to other actors. Therefore the effectiveness and overall performance of the acquisition will be influenced by where in the acquisition life cycle these actors are allocated. If constructive and knowledgeable actors are allocated in the early phases of the life cycle fewer errors will be propagated to later phases. It is therefore reasonable to assert that the variety of actors in the population, reflected by the variation in their attributes, may contribute either positively or negatively to the results of the acquisition.

For the purpose of this analysis constructive behaviour is characterised when the actor's Behavioural Style is between 0.24 and 0.42 (see Figure 5 in Section 5.7.4) and knowledgeable actors possess knowledge greater than 0.7 (where 0.0 and 1.0 represent respectively absence of knowledge and perfect knowledge). Table 24 illustrates the initial and final number of Constructive/Knowledgeable Team Member actors in the population for three representative cases (minimum, mean and maximum Effectiveness) extracted from the third test condition, (c) schedule constraint $P_c = 0.18$, shown in Table 23.

Table 24 – Fourth Hypothesis: Number of Constructive and Knowledgeable Actors

Scenario	Number of Constructive/Knowledgeable Team Member Actors		Achieved Effectiveness
	Initial	Final	
Non-Collaborative	32	29	0.53
	37	15	0.77
	37	67	0.93
Collaborative	37	102	0.78
	34	135	0.88
	40	234	0.97

The reduction in the number of Constructive/Knowledgeable actors in the course of the non-collaborative acquisition can be explained by the response of constructive actors to aggressive actions from leadership and management. This behaviour is explained by the dynamics of group interaction (Cooke & Szumal 1994). Constructive/ Knowledgeable actors may become passive, although still knowledgeable, and not so willing to share their knowledge and experience with others.

The increase in the number of Constructive/Knowledgeable actors in the course of the collaborative acquisition can be explained by the fact that less knowledgeable and experienced actors will learn from Constructive/Knowledgeable actors and will become more constructive in response to constructive actions from these actors as well as from managers, leaders and other actors. This behaviour is also explained by the dynamics of group interaction (Cooke & Szumal 1994).

Knowing the initial number of Constructive/Knowledgeable actors is not sufficient to determine what will be the effectiveness and overall performance of the acquisition. First, the definition of what Constructive/Knowledgeable implies: that the population may contain other actors that are close to the definition but missed the classification, or are within the boundary of the definition and could change their classification as the simulation progresses. Secondly, there is a random factor that could cause actors to change their behavioural style. An actor will not become knowledgeable and experienced by chance but could become passive, constructive or aggressive for no apparent reason. Therefore, although the characteristics of actors in the population and the consequent nature of the collaborative environment influence the final outcome, there is a component of uncertainty determined by lack of knowledge about the

population of actors, by the dynamics of the social system and by chance. The greater is the variety in the population of actors, the more difficult it is to predict the outcome.

Figure 27 shows the dynamics of performance and behaviour in the non-collaborative environment extracted from the third test condition, (c) schedule constraint, shown in Table 23.

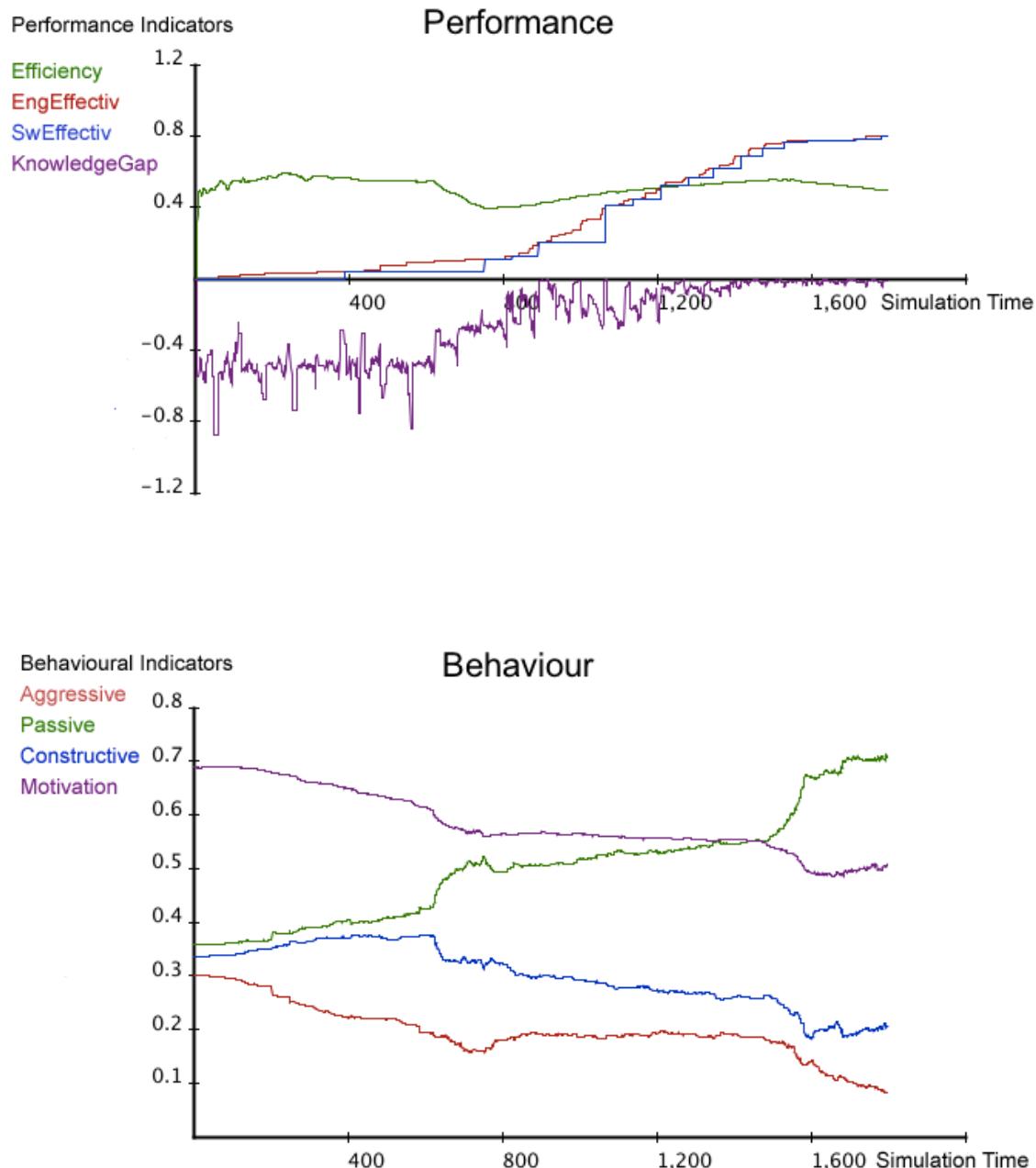


Figure 27 – Fourth Hypothesis: Non-Collaborative Performance and Behaviour

The sharp increase in passive behaviour seen in Figure 27 is most likely caused by the response of Team Members to aggressive behaviour from leadership and management. The increase in passive behaviour decreases the motivation, efficiency and the other two components of behaviour. Aggressiveness decreases because Team Member actors who were aggressive become passive. The component of aggressive behaviour that remains corresponds to Managers and Team Leader actors. It was observed that the abrupt increase of passivity may occur earlier or later in the course of the simulation and influences the overall effectiveness of the acquisition. The earlier the passive wave occurs the lower will be the effectiveness. Passivity reduces the willingness to cooperate and learn and increases time wasting. As the acquisition terminates when the target duration is reached there is no time to finish incomplete tasks and fix errors, resulting in a lower effectiveness of the solution.

Figure 28 shows the dynamics of performance and behaviour in the collaborative environment. In this environment all Managers, Team Leaders and Experts are constructive. Experts are Constructive/Knowledgeable actors. In addition some Team Members in the population may also be Constructive/Knowledgeable, as shown in Table 24.

There are two significant differences between the dynamics of the collaborative and non-collaborative environments. The first to be noticed is that constructive behaviour stands out throughout the acquisition, although it has been observed in other instances of the constructive scenario that there are occasions when passive and constructive behaviour have similar weights and sometimes the passive behaviour becomes more prominent at the end of the acquisition. The second difference is that the reduction of the knowledge gap is more accentuated in the constructive environment. As constructive behaviour promotes collaboration and learning it is particularly influential at the early stages of the acquisition because it reduces the *Chinese Whispers* effect (see Section 5.6.2). The earlier those constructive behaviours occur, and persist, the higher will be the effectiveness of the solution.

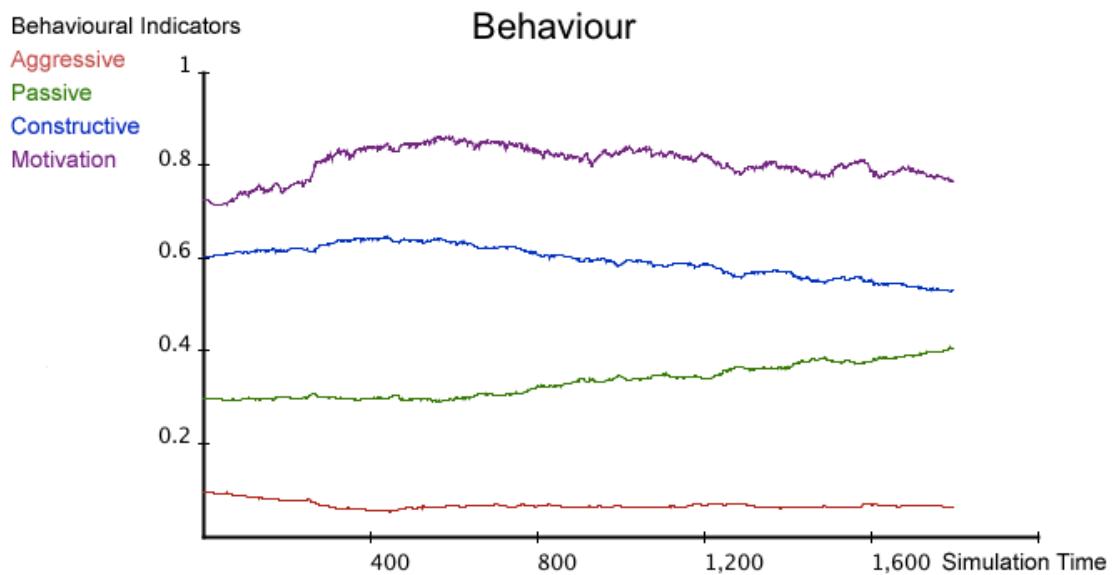
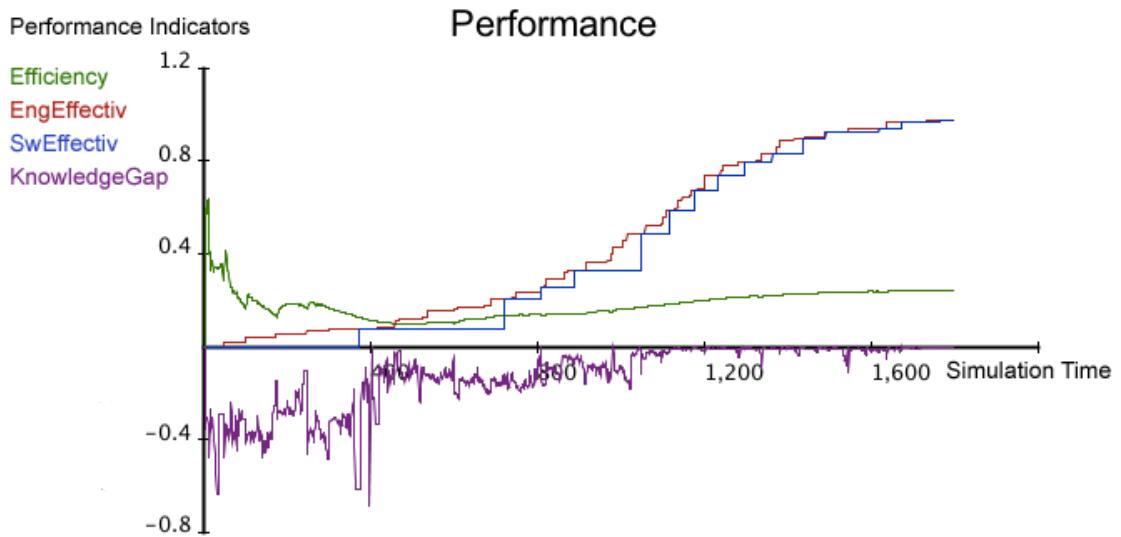


Figure 28 – Fourth Hypothesis: Collaborative Performance and Behaviour

The variation in actor attributes is shown in Table 25. For each of the two scenarios, collaborative and non-collaborative environments, the initial and final Mean (M) values and Standard Deviation (S) of Behavioural Style (B), Knowledge (K) and Experience (E) are shown for the lowest, mean and highest values of effectiveness reported by the simulation of the third test condition, (c) schedule constraint, shown in Table 23.

Table 25 – Fourth Hypothesis: Variation in Actor’s Attributes

Non-Collaborative Environment Scenario					
Performance	Actor’s Attributes	Initial		Final	
		Mean (M)	Std Dev (S)	Mean (M)	Std Dev (S)
$E = 0.53$	B	0.50	0.19	0.60	0.10
	K	0.50	0.19	0.96	0.12
	E	0.50	0.20	0.96	0.12
$E = 0.77$	B	0.51	0.20	0.62	0.08
	K	0.50	0.21	0.93	0.13
	E	0.51	0.19	0.95	0.11
$E = 0.93$	B	0.51	0.20	0.59	0.11
	K	0.50	0.21	0.97	0.11
	E	0.51	0.19	0.98	0.10
Collaborative Environment Scenario					
Performance	Actor’s Attributes	Initial		Final	
		Mean (M)	Std Dev (S)	Mean (M)	Std Dev (S)
$E = 0.78$	B	0.51	0.20	0.58	0.13
	K	0.50	0.20	0.99	0.05
	E	0.50	0.20	0.99	0.06
$E = 0.88$	B	0.50	0.20	0.57	0.13
	K	0.50	0.20	0.99	0.04
	E	0.50	0.20	0.99	0.06
$E = 0.97$	B	0.48	0.20	0.53	0.15
	K	0.49	0.20	0.99	0.06
	E	0.50	0.19	0.99	0.06

It can be observed in Table 25 that for both scenarios the average behaviour becomes Nice-Passive (Behavioural Style is between 0.50 and 0.58, see Figure 5), which is also observed in the graphics showing the dynamics of behaviour (Figure 27 and Figure 28). The higher effectiveness achieved by the collaborative environment is supported by higher levels of Knowledge (K) and Experience (E) and smaller variation.

Both environments cause a reduction in the variation of actor attributes, reflected by the reduction in the Standard Deviation (S) of all attributes (B, K and E), meaning that the actors become more uniform, though not identical. The collaborative environment performs better in reducing variation in knowledge and experience.

The simulation results confirmed that systemic structures that facilitate learning behaviours, named collaborative environments, reduce people variations. The simulation also confirmed that this characteristic is not unique to collaborative environments as non-collaborative environments also reduce people variations although less than the reduction observed in the presence of collaboration. The reduction of variation in people attributes is caused by the interaction between actors. Knowledge and experience increases for actors that collaborate and learn. Learning and

collaboration occur in both collaborative and non-collaborative environments but it is more frequent and intense in the first, thus the mean value of knowledge and experience is higher and more uniform in collaborative environments.

The simulation results also confirmed that collaborative environments increase the effectiveness of the solution, a condition that was also verified by testing the second hypothesis (see Section 6.4.3). It is important to observe that the increase in effectiveness is caused by the increase in the mean value of knowledge and experience of the actor population, and not by a reduction in the variation of actor attributes.

The simulation demonstrated that collaborative environments could improve schedule performance. This condition was verified when there were no imposed constraints of cost and schedule. Under this condition the solution emerged as a result of the capacity available in the population of actors, and the social system achieves what it is capable of achieving. It can be argued that under constructive leadership and without the pressure for meeting budgets and deadlines, actors develop constructive attitudes that facilitate learning and collaboration that improves the team performance. However, the knowledge and experience that exists in the system, combined with the capacity that the system has to learn and share knowledge, should be sufficient to execute the task within the imposed constraints of time and budget. Otherwise the system would fail in achieving the objectives intended for the system.

The cost of the acquisition in the collaborative environment was marginally higher because of the addition of 122 actors in the role of experts assigned to the teams. The variation in cost at the end of the simulation was similar for both collaborative and non-collaborative scenarios.

The simulation demonstrated that the effectiveness of the acquisition is very sensitive to variations in the initial population of actors. Understanding the variations in the population implies knowing what the actor's attributes are, how they change, and how they influence the outcomes of the acquisition. The uncertainty caused by not knowing what these variations are, reduces the ability to estimate, plan and predict the performance of the software-intensive acquisitions.

In conclusion the simulation confirmed the fourth hypothesis by verifying that systemic structures which facilitate learning behaviours reduce people variations and increase the

effectiveness of the solution, and may also improve the cost and schedule performance under certain conditions.

6.5. Summary

The Software-Intensive Acquisition Agent Based Model Simulation (SIAABMSim) has been presented together with a brief overview of the simulation process. The task configuration used for all simulation scenarios is based on the specification and development of the Super Seasprite SH-2G(A) Helicopter. The results of two simulations using earlier versions of SIAABMSim were reported as promising and led to the expansion of the tool and its application in this research.

The simulations presented in this chapter implemented a complex of models developed in Chapter 5 to represent the social and technical systems of software-intensive acquisitions and it was used to test and verify the four hypotheses formulated in earlier chapters of this thesis, confirming that:

- a. Lack of knowledge required to engineer the software-intensive solution increases cost, extends the schedule and reduces the effectiveness of the solution.
- b. Systemic structures that facilitate learning behaviours improve cost and schedule performance and the effectiveness of the solution of software-intensive acquisitions.
- c. Variations in people's knowledge, experience, personality and motivation cause variations in the effectiveness, cost and schedule of software-intensive acquisitions.
- d. Systemic structures that facilitate learning behaviours reduce people's variations and increase the effectiveness of the solution, and could improve schedule and cost performance.

Although numeric values from the simulation should not be taken as quantitative truth, the simulation results were sufficient to reproduce the qualitative behaviours that have been observed in large and complex acquisitions that experience schedule delays, cost overruns and solutions that do not always satisfy the user's needs. The simulation was

also sufficient to test ‘what-if’ conditions that could improve or worsen the overall performance of the acquisition.

The usefulness of the simulation results thus depends on the quality of models and parameters that configure the models in the simulation. Uncertainty about the fidelity of the models and accuracy of parameters propagates an uncertainty about the validity of results produced by the simulation. It is important to understand that this phenomenon is the reality of software-intensive acquisitions. Without knowing the attributes that drive the behaviour and performance of real people it is not possible to accurately estimate, plan and predict the results of the acquisitions. The uncertainty caused by not knowing what these attributes are is one of the causes of variations in cost, schedule and effectiveness of software-intensive acquisitions.

Chapter 7: Conclusions

7.1. Introduction

Software-intensive acquisition is the process of acquiring software-intensive products by specification, contract and eventual development, intended to operate as part of a solution to satisfy a need.

The acquisition of large software-intensive systems is a challenge for engineering and management. The history of failure in acquiring software-intensive systems within parameters of quality, cost and schedule makes the case for this research. While many large software-intensive acquisitions fail, a few succeed. Therefore, it is reasonable to assert that it is possible to succeed in software-intensive acquisitions.

This research intended to answer the most difficult question in the field of acquiring large software-intensive systems: *why is it so difficult to succeed ... and what can we do about it?* The difficulty in answering this question was expressed by Cobb's paradox: 'We know why they fail; we know how to prevent their failure; so, why do they still fail?' (Standish Group 1995)

The flaw that creates the paradox can be found in the way the paradox is stated. The paradox refers to 'they' as the 'projects', suggesting that software projects are living entities that can choose their own destiny. And yes, *they are*. Software projects and software-intensive acquisitions are social systems and are living entities that behave and adapt as result to people's actions and interactions. And yes, *they can*. Social systems are capable of doing what they do, including changing themselves and doing something different. Social systems define their own destiny, although the destiny may not be the explicit and intended aim for the system.

Social systems are capable of learning and adapting to new conditions. When the system is what it has to be to achieve the intended goals, success is a simple consequence that just happens. That is the case of simple software acquisitions where all is known and resources abound, but not typical of large software-intensive acquisitions where the gap of knowledge is large and allocated resources are an educated, and often unrealistic, guess. The system is not capable of managing itself to achieve a goal that is not its natural aim, and to align the aim of the system with

intended goals, a view from the outside is needed, provided by leadership and management. Without leadership and management the system will do what it does best, whether achieving the intended goal, getting close or moving far away from it, that could well cause a complete failure and self-destruction.

7.2. The Research

The review of success and failure of software-intensive acquisitions revealed that the problem lies in systemic causes, and Deming's System of Profound Knowledge was a natural choice to guide the research: appreciate the system; understand its variations; and developing theories about the system is essential for predicting future behaviour. As a social system, software-intensive acquisitions rely on how people behave as individuals and in teams and organisations, and some understanding of psychology and social sciences is essential.

The research problem was divided into three sub-problems: one raising questions about the behaviour of software-intensive acquisition as systems; the second raising questions about the variations in the system; and the third asking how to apply the findings of this research into a framework that would guide software-intensive acquisitions to success.

The review of the literature covered fields of systems, complexity theory, social behaviour and psychology, providing invaluable information to appreciate the software-intensive acquisition system and its variations.

Appreciate the System and Emergent Properties

The investigation about the system revealed that software-intensive acquisitions are socio-technical complex adaptive systems. The system has its own implicit goals that are not always aligned with the intended and explicit goals for the system. Success depends on properties that emerge when the system is in operation and therefore cannot be satisfactorily predicted before the system comes to existence. Knowledge, quality and competency are interconnected and are desirable emergent properties that contribute to the system achieving the intended goals. The effectiveness of the solution will depend upon how people apply their knowledge and experience, and collaborate to create new and collective knowledge, guiding the intrinsic aims of the system to the intended aims for the system.

Two hypotheses were formulated about the behaviour of the system: the first about the impact of lack of knowledge in the success of the acquisition; and the second about how systemic structures that facilitate learning behaviours would help the acquisition to achieve the intended goal.

Understanding Variations in the System

The investigation about variations in the software-intensive acquisitions system indicated that the observed variations in cost, schedule and quality are consequences of variations in the ability of people to perform their tasks. Variations in people's knowledge, experience, personality and motivation influence variations in cost, schedule and the quality of the software-intensive solution.

Two hypotheses were formulated about variations in the system: one suggests that variations in people's knowledge, experience, personality and motivation cause variations in the effectiveness, cost and schedule of software-intensive acquisitions; and the other suggests that systemic structures that facilitate learning behaviours can reduce people's variations and variations in cost, schedule and effectiveness of the software-intensive solution.

A Model of Software-Intensive Acquisitions

A model of software-intensive acquisitions was developed with the objective of testing in a computer-simulated environment the four hypotheses about the system and its variations. The model was designed to include relevant aspects of the real system: a socio-technical complex adaptive system where people apply their attributes of knowledge, experience, personality and motivation to collaborate and perform tasks to develop a software-intensive solution for a need.

The agent-based modelling (ABM) approach was chosen for the model as it provides the required properties for modelling cognitive adaptive agents and non-cognitive agents that interact in a complex adaptive system. ABM provides the heterogeneity needed for modelling variations in people and artefacts represented respectively by cognitive and non-cognitive agents.

The Software-Intensive Acquisition Agent-Based Model (SIAABM) adopts the ACTS theory proposing that organisations are collections of intelligent agents cognitively restricted, task oriented, and socially situated. SIAABM includes a task model that

represents the technical system that is the subject of the acquisition; a cognitive agent model and a social model representing the social system. Cognitive agents are named actors and non-cognitive agents, artefacts.

The task model comprises products, artefacts, transformations and tasks. Artefacts are components of products developed during the engineering process. Input products are transformed into output products by tasks assigned to actors. The quality of the output product depends upon how well the actors perform the task.

Actors are defined by their attributes of role, behavioural style, knowledge, experience and motivation. Actors adapt and change their attributes as they interact with other actors, with the environment and perform tasks assigned to them. Actors are grouped in teams, departments and organisations and interact in accordance with their professional roles and informal relationships.

SIAABMSim

SIAABM was applied to develop a simulation tool, Software-Intensive Acquisition Agent-Based Model and Simulation (SIAABMSim) as an application of Repast, a Java agent-based modelling and simulation toolkit. SIAABMSim provides a simulation environment that is sufficient to test the four hypotheses formulated by this research. The results of the simulation tests were used to answer the research sub-problems.

7.3. Answer to the Research Sub-Problems

The answers to the first and second research sub-problems come from the appreciation of the software-intensive acquisition as a system, and an understanding of the variations in the system. The confirmation of the hypotheses through computer simulation substantiates the answers provided by this research.

7.3.1. Answer to Sub-Problem 1

The first research sub-problem addresses the systemic aspect of software-intensive acquisitions, formulated by the question:

What can be learned about the software-intensive acquisition system that would help to understand the system's ability in achieving its intended goal?

The investigation of the software-intensive acquisition system confirmed that success or failure is a consequence of many factors that determine the systemic structure of the system. The system does what it is capable of doing and often what the system does is not what is needed to achieve the intended goals.

The test of the first and second hypotheses in a computer-simulated environment confirmed that lack of knowledge and experience required to engineer the software-intensive solution increases cost, extends schedule and reduces the effectiveness of the solution; and, that systemic structures that facilitate learning behaviours improve cost and schedule performance and the effectiveness of the solution.

Knowledge and behaviour are two important aspects for the success of software-intensive acquisitions.

7.3.2. Answer to Sub-Problem 2

The second research sub-problem addresses the variations in the software-intensive acquisition system, and it is formulated by the question:

What are the causes of variations in the software development component of software-intensive acquisitions and how do these variations affect the ability to achieve the established parameters of cost, schedule and quality?

The investigation of the variations in the software-intensive acquisition system indicated that the observed variations of cost, schedule and quality are caused by variations in people.

The test of the third and fourth hypotheses in a computer-simulated environment confirmed that variations in people's knowledge, experience, personality and motivation are the root causes of variations in cost, schedule and quality of the software-intensive solution; and, that systemic structures that facilitate learning behaviours reduce people's variations and the observed variations of cost, schedule and effectiveness of the solution of the software-intensive acquisitions.

7.4. Answer to the Research Problem

The fundamental problem addressed by this research is formulated by the question:

Why is it so difficult to succeed in large software-intensive acquisitions and in what ways can the software development component of software-intensive acquisitions be adequately structured and managed to effectively achieve the established parameters of cost, schedule and quality?

The difficulty to succeed in large software-intensive acquisition is created by poorly understanding the acquisition activity as a system, i.e. poorly understanding what is the intended aim for the system, and what is the intrinsic aim of the system.

Software-intensive acquisitions fail when the *intended aim for the system* is not aligned with the *intrinsic aim of the system*, and *the intrinsic aim of the system is what the system really does*.

The condition to succeed in achieving the established parameters of cost, schedule and quality is what is needed to align both aims, by changing the intended aim for the system, or by changing the implicit aim of the system, or by changing both aims.

The difficulty is in how to align the two aims: first, in identifying what is the intrinsic aim of the system; second, to assess whether the aim of the system can be aligned to the intended aim; and finally, how to create the conditions that would bring the implicit aim of the system together with the intended aim for the system.

The *intended aim for the system* is determined by how people and organisations understand what is documented in contracts, specifications and plans. The *intended aim for the system* is therefore the social understanding of the *need*, the required *solution*, the plans to complete the task and the reward paid to achieve the intended aim.

The *intrinsic aim of the system* is determined by the systemic structure that makes the system do what it does as the result of how people as individuals and in organisations apply their knowledge, experience and motivations, and interact to achieve the aim of the system.

To bring the two aims together is a matter of understanding the system and changing the prevailing management style, as suggested by Deming, and a framework for software-intensive acquisitions, as a step towards a theory, can help.

7.5. The Need for a Framework

The theory of knowledge introduced by the System of Profound Knowledge (Deming 2000) states that knowledge is not sufficient to predict, and that only when knowledge is transformed into a theory that prediction is possible. The theory of knowledge also states that a theory may not be complete or perfect, as long as it yields useful predictions. If in practice prediction is proved to be incorrect, the theory should be revised.

This research combined and developed knowledge of software-intensive acquisitions that could be transformed into a theory. This research acknowledges that knowledge acquired through a hypothetical simulation is not unquestionable evidence of truth. The simulation results, however, confirms the kinds of behaviours observed in real software-intensive acquisitions that experience delays and cost overruns. Therefore, the findings of this research can contribute towards the development of a theory, until such time as they are disproved, whereupon it would be revised.

Rather than to be so bold as to claim a theory where there is a clear and falsifiable hypothesis, this research applies its findings to formulate a *framework*, as a structure of principles, assumptions, concepts, values, and practices that constitutes a way of viewing reality of software-intensive acquisitions. After validation in practice the framework would lead to a detailed theory of software-intensive acquisitions.

The framework proposed by this research is the answer to the third research sub-problem formulated by the question:

How can a better understanding of the software-intensive acquisition system and its variations help to formulate a framework to improve prediction and the chances of success?

Decision makers, managers and leaders of software-intensive acquisitions often try to control a system that cannot be controlled. Rather, the system should be *steered* towards

the alignment of both intended and intrinsic aims, and a *framework for steering software-intensive acquisitions to success* can guide this task.

7.6. A Framework for Steering Software-Intensive Acquisitions to Success

The essence of this research is captured by two statements about what a system truly is:

System is a network of interdependent components that work together to try to accomplish the aim of the system (Deming 2000 p. 50).

The purpose of a system is what it does (Beer 2002 p. 218).

‘*The aim of the system*’ in Deming’s definition is in fact the ‘*intended aim for the system*’ while Beer’s statement refers to the ‘*intrinsic aim of the system*’. When these two aims are not aligned the software-intensive acquisition is destined to fail.

This research highlights awareness about the importance of understanding software-intensive acquisitions as socio-technical systems and their intrinsic complexity, moving the current understanding about software-intensive acquisitions from the highest level of ignorance, Fourth level of Ignorance (4OI), *there is no knowledge about the Five Orders of Ignorance*, to the Third Order of Ignorance (3OI), i.e. *some of the required knowledge is available, but not all; the kind of knowledge to be acquired is unknown, and there is no suitable process of discovering in place* (Armour 2004; see Table 5 in Section 2.4.7).

The results from the research are encapsulated in a *framework for steering*, rather than attempting to control, *software-intensive acquisitions to success*. This framework can be applied as a process to improve understanding about some of the aspects of software-intensive acquisitions to the Second Order of Ignorance (2OI), i.e. *some of the required knowledge is available, but not all; the kind of knowledge to be acquired is unknown, but there is a suitable process of discovering in place* (Armour 2004; see Table 5 in Section 2.4.7).

Tom DeMarco supports the idea of steering software development without controlling it (DeMarco 2009). DeMarco, who once advocated measurement and control of software development, acknowledged that not all in software development could be measured, much less controlled. Without recommending specifically agile methods for software

development DeMarco suggests that a most appropriate approach to manage software development is ‘one that might well steer the team toward agile methods, at least toward the incremental aspect of the agile school’ (DeMarco 2009 p. 95).

Watts Humphrey, in what could have been his last interview⁴⁴, suggested that software development management should be treated as ‘a continuous learning process, as a leading process, and not as a directional process’ (Humphrey 2010b p. 8).

A *framework for steering software-intensive acquisitions to success* should be made simple, as long as it conveys knowledge that allows predictions⁴⁵.

Simple statements, if knowledge is our object, are to be prized more highly than the less simple ones because they tell us more; because their empirical content is greater; and because they are better testable. (Popper 1959 p. 128)

Two frameworks referenced by this research, the System of Profound Knowledge and the Framework for Harnessing Complexity, convey invaluable knowledge through simple statements. Wisdom comes when the combined meaning of simple statements is realised and applied to achieve an objective.

This research showed that without the required experience and knowledge software-intensive acquisitions are likely to experience schedule delays and cost overruns, and deliver solutions that do not satisfy the user’s needs. Therefore, organisational knowledge is essential for the success of software-intensive acquisitions. Organisational knowledge starts with the individuals in the organisation (Nonaka 1994, in Section 2.4.7), and the members of the organisation ought to be motivated to share knowledge and experiences and learn from each other (Xio-qing & Nan 2010, in Section 2.4.7).

This research also showed that leadership that motivates the members of the organisation towards constructive-learning behaviours are central aspects of successful software-intensive acquisitions, confirming Lawson’s observation that successful projects take social aspects into consideration, and are focused on the character and behaviour of the people involved in the project (Lawson 2005 in Section 2.5).

⁴⁴ Watts S. Humphrey died in October 2010 at the age of 83 (SEI 2010).

⁴⁵ Paraphrasing an unconfirmed quote attributed to Albert Einstein: ‘Everything should be made as simple as possible, but no simpler.’

Supported by Deming, Beer, DeMarco and Humphrey and adopting the simplicity principle suggested by Popper, the proposed *framework for steering software-intensive acquisitions to success* contains one principle about the *system and its aims* and three management processes of *knowledge, behaviour* and *motivation*, to align the aims of, and for, the system.

7.6.1. The System and its Aims

The answer to the research problem (Section 7.4) led to the conclusion that software-intensive acquisitions succeed when the software-intensive acquisition system presents the conditions that are required for the system to achieve the intended goals.

The principle that guides the *framework for steering software-intensive acquisitions to success* is stated as follows:

*Software-intensive acquisitions succeed when the **intended aim for the system** is aligned with the **intrinsic aim of the system**.*

*The **intended aim for the system** is what is expected the system will be able to do.*

*The **intrinsic aim of the system** is what the system really does.*

The intended aim for the system is determined by the development of an acceptable solution for an expressed need within parameters of cost schedule and quality. The intrinsic aim of the system is determined by what the system is capable of achieving in accordance with the capacity of people and organisations, in the form of their knowledge, experience and skills within the context of the work to be performed, combined with their motivations and behavioural styles. The gap between what the system is and what it should be to achieve the intended goals must be reduced to acceptable levels for a successful acquisition.

The three processes that follow can be applied to align the intrinsic aim of the software-intensive acquisition system with the intended aim for the system. It needs to be observed that the software-intensive acquisition system has its intrinsic limits that are constrained by the physical and mental capacity of the people in the system as well as by the imposed cost and schedule. Therefore, the alignment may not be possible if it would require stretching the system constraints beyond realistic and acceptable levels.

7.6.2. The Process of Knowledge Management

The term *knowledge* broadly refers to knowledge, experience, expertise and skills. There are two distinctive perspectives about knowledge management in software-intensive acquisitions. First is *management knowledge*, as the knowledge needed to understand the system, its variations, constraints and aims. *Management knowledge* includes theories that allow prediction about the behaviour of the system, as suggested by the System of Profound Knowledge, and brings an understanding of what are the *intended aim for the system* and the *intrinsic aim of the system*. Managers, leaders and decision makers in the acquisition process must possess the required *management knowledge* to ascertain what the system can and cannot do, and for making the necessary changes in the system to steer it towards the intended goals. The *framework for steering software-intensive acquisitions to success* is a contribution to *management knowledge*.

The second aspect of knowledge in software-intensive acquisitions is the *engineering knowledge*. The process of engineering software-intensive solutions calls for three domains of knowledge: domain knowledge to find a solution for an expressed need; knowledge of software technology to translate the solution into software code; and knowledge about learning processes and how to work in a collaborative team (Armour 2004; Chan, Jiang & Klein 2008).

It is expected that software-intensive acquisitions start with a large knowledge gap, as the difference between the available knowledge and experience and what is needed to engineer and develop the software-intensive solution. The Process of Knowledge Management applies the concepts from the Five Orders of Ignorance as a model for understanding the process of learning (Armour 2004 p. 138) introduced in Section 2.4.7, Table 5. The aim of the Process of Knowledge Management is thus to move the software-intensive acquisition from a higher Order of Ignorance (OI) – Fourth OI (Meta Ignorance) or Third OI (Lack of Process) – to at least the Second OI (Lack of Awareness) where there is a process to discover the unknown unknowns. At the Third OI the Process of Knowledge Management is meant to create the processes to discover what is unknown (Armour 2004).

This research proposes the adoption of specific Knowledge Management Plan (KMP) to document the strategy and processes to reduce the existing knowledge gap. The original

refereed paper published by this research proposing the adoption of a specific KMP is included in Appendix E. The emphasis on the plan being ‘specific’ is to make a clear distinction that the aim is to address specific aspects of knowledge management for the software-intensive acquisition and not the philosophical nature of knowledge management programs. The KPM identifies the knowledge required for the acquisition; the knowledge available in the software-intensive acquisition system and where the knowledge is, i.e. ‘who knows what’; the known gap of knowledge; the process to discover ‘what we don’t know that we don’t know’ and developing a plan to implement the strategy to reduce the knowledge gap.

The KPM is a learning plan tailored to the software-intensive acquisition enterprise and defines the learning activities to meet the learning objectives. Learning activities create organisational knowledge by increasing the knowledge of individuals (Nonaka 1994 in Section 2.4.7), whether by transforming explicit knowledge into tacit knowledge through specific training and self-paced learning, by sharing tacit knowledge and experience through collaboration and mentoring, or by exploration activities as a process of discovery.

The KPM also includes the cost and time required to fill the gap of knowledge and the risks involved if the gap is not resolved. The specific KPM makes explicit the real cost and time to meet the knowledge requirements for the success of the software-intensive acquisition and provides the information for a realistic assessment against imposed constraints of cost and schedule.

The *process of knowledge management* is consistent with the recommendations from the GAO report on Defence Acquisitions of Selected Weapon Programs 2009 (GAO 2009) recommending a knowledge-based acquisition approach to address the pervasive problems of cost and schedule variation.

7.6.3. The Process of Behaviour Management

The objective of the *process of behaviour management* is to improve the software-intensive acquisition outcomes through the way that people interact in groups and organisations. The *process of behaviour management* aims to maximise constructive behaviours, promote cooperation and increase the effectiveness of the *process of knowledge management*.

The *process of behaviour management* applies principles from the dynamics of group interaction and the impact of group interaction style on problem-solving (Cooke & Szumal 1994) discussed in Section 2.4.6. Software-intensive acquisitions are a form of problem-solving group, and according to Cooke and Szumal, problem-solving groups tend to increase their effectiveness through constructive behaviours that promote cooperation and learning. The dynamic of group interaction tells that constructive behaviour promotes constructive behaviour, and aggressive behaviour suppresses constructive and promotes passive behaviours.

Group members with more power, as in management or leadership roles, or with stronger personality styles are likely to influence other members in the group (Cooke & Szumal 1994). As discussed in Section 2.4.6, autocratic leaders can be correlated with power-aggressive/non-constructive behaviour, and transformational leaders demonstrate constructive behaviour. While constructive behaviour is likely to impel cooperation, non-constructive interaction styles can undermine attempts at knowledge sharing and achieving knowledge management goals (Balthazard & Cooke 2004).

This research proposes the adoption of a specific Behaviour Management Plan (BMP) to develop strategies and set targets for managing stakeholders, individual and group behaviour and organisational culture. The publication of this research in the original refereed paper, proposing the adoption of a specific BMP, is included in Appendix E.

Stakeholder management identifies people and organisations capable of influencing the acquisition. Stakeholder management attempts to ascertain the stakeholder's interests and their likely behaviour, and to identify project risks that could come from stakeholders capable of swaying the acquisition negatively. Stakeholders include engineers and developers; team leaders and managers; customers and users; and observers with a high stake in the outcomes of the acquisition. Stakeholder management is essential for effective project management (Bourne & Walker 2003). The Stakeholder Circle (Bourne & Walker 2006) is an example of a stakeholder management tool used to identify, prioritise and develop a stakeholder engagement strategy that better suits the project and objectives of the business enterprise.

The Life Styles Inventory (LSI) (Lafferty 1973, in Cooke & Rouseau 1983 in Section 2.4.4) is a behaviour management tool that uses questioners to assess the behaviour profile of individual members of the organisation. The assessment is in two parts: a self-

assessment (LSI Level I), and an assessment performed by others (LSI Level II) producing a classification as the person is perceived by peers, subordinates and superiors. The intent of the LSI as an individual behavioural management tool is to make people aware of their own behavioural style as self-perceived and as perceived by others and motivate them to modify undesirable behaviours and strengthen behaviours that are beneficial for the group and the organisation. By changing individual behaviour, group behaviour and organisational culture can be shaped.

Group behavioural style and organisational culture can be assessed with questioner based tools like the Group Style Inventory (GSI) (Cooke & Lafferty in Cooke & Szumal 1994) and the Organisational Culture Inventory (OCI) (Cooke & Szumal 2000). LSI, GSI and OCI can be used to assess the effectiveness of the *process of behaviour management*, by comparing the results of periodic LSI, GSI and OCI assessments against the behaviour and cultural transformation targets defined in the BMP.

When applied to software-intensive projects, behaviour management should be extended to all organisations (i.e. user, customer and providers) involved in the software-intensive acquisition to be able to produce effective results. Every organisation in the acquisition is a component of the software-intensive acquisition system that should be aligned towards the intended aims for the system. Only with consistent constructive attitudes and the right balance between transactional and transformational leadership, i.e. task-focused versus people-focused leadership (Bass 1990; see Leadership Styles in Section 2.4.6) it is possible to influence the software-intensive acquisition system to align both intrinsic and intended aims.

7.6.4. The Process of Motivation Management

The *process of motivation management* aims to give incentives to individuals and organisations to behave and act in ways that steer the software-intensive acquisition towards the intended objectives.

The theory of motivation was discussed in Section 2.4.2. At the individual level, *motivation* is the intervening process or internal state of a person that drives their behaviour or impels them into action (Reber & Reber 2001). The success of the acquisition depends on the work of talented and motivated managers and engineers, and what motivates one does not necessarily motivate the other (McConnell 1996). While managers tend to be motivated by the challenge of meeting contractual obligations,

developers are likely to be motivated by the technical challenge of engineering a quality solution that requires creativity, learning and applying new technologies (Linberg 1999; McConnell, 1996; Procaccino, Verner & Lorenzet 2006). The different nature of engineering and management tasks, and conflicting motivations, creates tension between engineering and management (Aslaksen 1996). People are differently motivated by intrinsic and extrinsic factors (Ryan & Deci 2000 in Section 2.4.2) and managers should be aware of an individual's motivation factors to be able encourage their employees to learn from each other (Xio-qing & Nan 2010; Qinxuan & Yingting 2010 in Section 2.4.7). Finding the individual motivation strategy that best fits a particular software-intensive acquisition system is part of the *process of motivation management*.

At an organisational level, motivation is driven by extrinsic motivation established by contracts that specify responsibilities, rewards and liquidated damages in case the contract is not fulfilled. Typically software-intensive acquisitions involve a number of contractors and sub-contractors each with their own capabilities, interests and motivations. The contracting model defining responsibilities, rewards and liquidated damages that fit the acquisition as a system of organisations is therefore important to success. Deficiencies in contracts have contributed to the total failure of software-intensive acquisitions, as happened with the Super Seasprite Helicopter programme (Ferguson & Blenkin 2008 in Section 3.7.4). Software-intensive acquisitions often adopt fixed-price contracts that create an illusion that the customer is protected. The customer will always lose if the contractor fails to deliver a critical capability on time. Again, the Super Seasprite Helicopter programme is an example of this nature (Ferguson & Blenkin 2008). Liquidated damages are not sufficient to compensate the customer for not having the capability in operation. Contracting models that adequately and consistently fit the reality and complexity of software-intensive acquisitions constitute a challenge yet to be conquered.

7.6.5. Emergent Results

There is nothing new in applying a ‘systems view’ and processes to manage knowledge and behaviour in large projects. The case of the development of the on-board software for the Boeing 777 (DAF 2000 Appendix Q), discussed in Section 3.6.1, shows the management understanding of the system and the application of knowledge and behavioural management, although not explicitly reported. The fact is that the consistent

application of combined processes led to emergent results that went beyond what each process would achieve in isolation.

Knowledge, behaviour and motivation influence each other. Figure 29 shows the dynamics of motivation, behaviour and action created by the three processes of the proposed *framework for steering software-intensive acquisitions to success*. Lack of knowledge extends schedules and increases costs and often causes management aggressive intervention that is unlikely to address the root cause of the problem. Instead, aggressive attitudes from management are often answered with passivity and demotivation, and worsen the situation. Constructive behaviour fosters learning and collaboration and contributes to reduce existing gaps of knowledge, which in turn will lead to better results increasing the motivation of individuals, teams, departments, organisations and the acquisition as a whole.

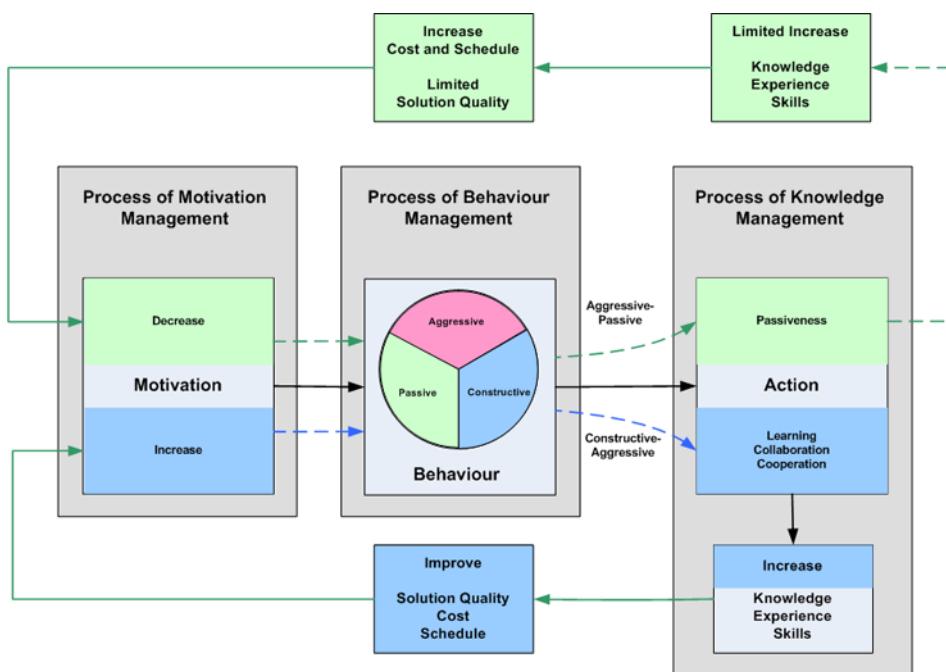


Figure 29 – Motivation, Behaviour and Action

Embracing knowledge, behaviour and motivation management through realistic plans and their effective application is an important element of socio-organisational design that can improve performance and the chances of success of the software-intensive acquisition. The integrated and consistent application of the three processes will foster the development of the desirable emergent properties (see Section 3.6) and steer the software-intensive acquisition system towards the intended goals.

7.7. Limitations of this Research

This research focussed on the engineering process of the software-intensive acquisition system and how people and social structures influence cost, schedule and the quality of the solution. Given the complex nature of the software-intensive acquisition system this research acknowledges that there may be many other known and unknown factors that could influence the system that have not been addressed. Some of the known factors include the influence of politics and external stakeholders; the influence of contracting and reward models; and the implications of organisational structures associated with functional versus matrix models. These and other factors could be the subject of further research.

Hypotheses were tested by computer simulation using hypothetical data, and the models for cognitive agents and social dynamics were based on social theories that are qualitative in nature. Therefore, simulation results produced by this research should not be taken as quantitative evidence of any specific software-intensive acquisition.

The model and simulation developed and used by this research were considered to be valid for the purpose of testing the hypotheses on the basis that the simulation results reflected the kind of qualitative behaviours and outcomes that have been empirically observed in real software-intensive acquisitions. However, this research did not perform a formal validation of the model and simulation results. Validation should occur by applying SIAABM and SIAABMSim to a real software-intensive acquisition for which reliable data is available and the models could be better calibrated.

The proposed *framework for steering software-intensive acquisitions to success* is the first step towards a theory of software-intensive acquisitions and it is yet to be tested in practice. The application of the framework in a real acquisition is the best way to test its effectiveness, as well as to revise and refine the theory behind the framework.

7.8. Future Work

This research suggests three paths for future work: (1) validate SIAABM and SIAABMSim against real software-intensive acquisitions; (2) explore existing features, refine and expand the capabilities of SIAABM and SIAABMSim; and (3) validate the effectiveness of the proposed *framework for steering software-intensive acquisitions to success*.

7.8.1. Validate SIAABM and SIAABMSim

The validation of SIAABM and SIAABMSim against a real software-intensive acquisition imposes interesting challenges. The first difficulty is in obtaining the data to instantiate the model. Validation against past acquisitions would most likely face the fact that quality data that is complete and accurate will not be available. Gaps from non-existent data would have to be filled with assumptions, in a similar manner to this research. The validation of assumptions and results from the simulation would then proceed through discussions with stakeholders that participated in the acquisition.

Alternatively, SIAABM and SIAABMSim could be validated against an on-going software-intensive acquisition, preferably from start-to-finish. In this case, the required data would be collected in the course of the acquisition and prepared to be used in the simulation. Data collection and preparation for simulation is a vast field for research that would combine psychology and social sciences with modelling and simulation.

7.8.2. Explore, Refine and Expand SIAABM and SIAABMSim

SIAABM and SIAABMSim have many features that were not extensively used in this research. These features could be explored to investigate other aspects of interest that have been not addressed by this research, and applied not only to software-intensive acquisitions but also to other kinds of engineering projects and socio-technical systems in general.

The SIAABMSim configuration files included in Appendix A show a variety of parameters that can be configured to investigate, for example, the influence of formal and informal interactions in learning and behaviour; the influence of implicit and explicit rewards as drivers of intrinsic and extrinsic motivations; the influence of feedback learning in the effectiveness of the solution; how delaying reward and punishment influence the acquisition outcomes; among many other possibilities.

SIAABMSim also provides a mechanism for configuring individual actors and modifying actors during the simulation. This feature permits the investigation of the influence of individual actors, and the effect of replacing actors in the course of the acquisition as a result of staff turnaround. The tool also allows the configuration of teams, departments and organisations with specific knowledge, experience and behaviour profiles, providing a rich environment to investigate how differences between

organisations in the acquisition system would influence cost, schedule and quality, e.g. an inexperienced and aggressive customer working with a knowledgeable, experienced and constructive provider, and any other combination.

Comparing the simulation results against what is realistically expected, and then adjusting the rates of change and probability of events, included as configuration parameters, could be used to calibrate the model. In case changing configuration parameters is not sufficient to produce the desired result, the algorithms implemented in the source code could be modified to refine the model.

Finally, SIAABM and SIAABMSim could be expanded to include other types of artefacts, actors, interactions, tasks, dependencies and social structures as necessary to investigate many other aspects of interest in software-intensive acquisitions and socio-technical systems that have been not addressed by this research. Section 5.7.2 suggested, for example, that the model could be expanded to explore the influence of internal and external stakeholders, which would require the creation of observer actors, their stake profile and interactions with other actors.

7.8.3. Validate the Effectiveness of the Proposed Framework

The validation of the effectiveness of the proposed *framework for steering software-intensive acquisitions to success* needs to occur by applying the framework in a real software-intensive acquisition from start-to-finish. The application of the framework should be supported by a plan to establish how the framework would be implemented. Each of the three processes in the framework should be sufficiently detailed and people involved in applying the framework should receive appropriate training. The application of the framework could be combined with modelling and simulation. The project could become an integrated research programme involving the three future work paths here proposed applied to a specific software-intensive acquisition.

It would also be interesting to compare the outcomes of two software-intensive acquisitions: one that would apply the *framework for steering software-intensive acquisitions to success* supported by the modelling and simulation, and the other executed in accordance with a traditional management approach. This way, and with a deliberate and well designed experiment, the effectiveness of the *framework* could be tested.

7.9. Final Remarks

This research confirmed what is intuitively known and a few dare to say. Software-intensive acquisitions may or may not have what it takes to succeed. Management can influence success as long as the system has the capacity to adapt to the required conditions. Success is not a certain consequence of management will; rather, success is a consequence of the nature of the system. By understanding the software-intensive acquisition system managers would be able to assess the system and create realistic conditions to change the system to reflect what is required to achieve the intended success, respecting the fact that the system has its own limits for change. The framework proposed by this research can be applied to improve our understanding about software-intensive acquisitions.

Modelling and simulation is a rich learning environment. The benefits of simulation are not only the results the simulation provides. By investigating what is required to model or simulate the system, a better understanding of the system itself is gained. The process of constructing a simulation of complex systems can be as beneficial as, if not more than, the results that arise from the simulation itself.

Decision makers, managers and leaders in software-intensive acquisitions would acquire significant knowledge by exploring ‘what if’ scenarios with the simulation tool developed by this research; and would benefit even more from the experience of discovering what is needed to model the system they are responsible for managing and leading to success.

Bibliography

- Akao, Y 1990, *Quality Function Deployment: Integrating Customer Requirements into Product Design*, Productivity Press, Portland, OR.
- ANAO 2008, *2007-08 Major Projects Report – Defence Materiel Organisation*, 27 Nov. 2008, Australian National Audit Office, ANAO Report No. 9 2008-09, Commonwealth of Australia, Canberra, ACT.
- ANAO 2010a, *Acceptance into Service of Navy Capability*, 28 June 2010, Australian National Audit Office, ANAO Report No. 57 2010-11, Commonwealth of Australia, Canberra, ACT.
- ANAO 2010b, *2009-10 Major Projects Report – Defence Materiel Organisation*, 30 Nov. 2010, Australian National Audit Office, ANAO Report No. 17 2010-11, Commonwealth of Australia, Canberra, ACT.
- Armour, P 2000, ‘The Five Orders of Ignorance’, *Communications of the ACM*, vol. 43, no. 10, pp. 17-20.
- Armour, P 2001, ‘The Laws of Software Process’, *Communications of the ACM*, vol. 44, no. 1, pp. 15-17.
- Armour, P 2004, *The Laws of Software Process: A new Model for the Production and Management of Software*, Auerbach Publications, Boca Raton, FL.
- Ashby, WR 1957, *An Introduction to Cybernetics*, Chapman and Hall, London.
- Aslaksen, EW 1996, *The Changing Nature of Engineering*, McGraw-Hill, Sydney.
- Axelrod, R & Cohen, MD 2000, *Harnessing Complexity, Organisational Implications of a Scientific Frontier*, Basic Books, New York, NY.
- Bar-Yam, Y 2003, ‘When Systems Engineering Fails – Towards Complex Systems Engineering’, *International Conference on Systems, Man & Cybernetics 2003*, vol. 2, pp. 2021-2028, IEEE Press, Piscataway, NJ.

- Balthazard, PA & Cooke, RA, 2004, ‘Organizational Culture and Knowledge Management Success: Assessing The Behavior–Performance Continuum’, *Proceedings of the 37th Hawaii International Conference on System Sciences – 2004*, Big Island, HI.
- Bass, BM, 1990, ‘From Transactional to Transformational Leadership: Learning to Share the Vision’, *Organizational Dynamics*, vol. 18, no. 3, pp. 19-31.
- Beer, S 1972, *The Brain of the Firm*, Allen Lane The Penguin Press, London.
- Beer, S 1979, *The Heart of Enterprise*, John Wiley & Sons Inc., New York, NY.
- Beer, S 2002, ‘What is Cybernetics’, *Kybernetes: The International Journal of Systems and Cybernetics*, vol. 31, no. 2, pp. 209–219.
- Berson, Y & Linton, J 2003, ‘An Examination of the Relationships Between Leadership Style, Quality and Employee Satisfaction n R&D Environments’, *Engineering Management Conference, 2003, IEMC '03, Managing Technologically Driven Organizations: The Human Side of Innovation and Change*, 2-4 Nov. 2003, Albany, NY.
- Bertalanffy, L 1969, *General System Theory: Foundations, Development, Applications*, George Braziller, Inc., New York, NY.
- Bishop, M 2007, ‘Speech: Defence Procurement’, *The Senate Matters of Public Interest, Parliamentary Debates*, Commonwealth of Australia, 13 June 2007, Canberra, ACT.
- Boghossian, ZJ 2002, *An investigation into the critical factors of software development process, time and quality*, EdD dissertation, Pepperdine University, UMI Dissertation Services, Ann Arbor, MI.
- Boehm, B 1988, ‘A Spiral Model of Software Development and Enhancement’, *IEEE Computer*, vol. 21, no. 5, pp. 61-72.
- Boehm, B 1991, ‘Software Risk Management: Principles and Practices’, *IEEE Software*, vol. 8, no 1, pp. 32-41.

Boehm, B, Brown, W & Turner, R 2005, 'Spiral Development of Software-Intensive Systems', *ICSE '05: Proceedings of the 27th international conference on Software engineering*, pp. 706-707.

Boehm, B, Valerdi, R & Honour, E 2008, 'The ROI of systems engineering: Some quantitative results for software-intensive systems', *Systems Engineering*, vol. 11, no. 3, pp. 221-234, Wiley Periodicals, Inc..

Bourne, L & Walker, DHT 2003, 'Tapping the Power Lines: A 3rd Dimension of Project Management Beyond Leading and Managing', *7th Australian International Performance Management Symposium*, Canberra, ACT.

Bourne, L & Walker, DHT 2006, 'Using a Visualising Tool to Study Stakeholder Influence', *The Project Management Journal*, 2006, vol. 37, no. 1.

Briggs, I & Myers, PB 1980, *Gifts Differing: Understanding Personality Type*, Davies-Black Publishing, Mountain View, CA.

Brooks, FP 1995, *The Mythical Man-Month: Essays on Software Engineering*, 20th Anniversary Edition, 1995, Addison-Wesley Publishing Company, USA.

Carley, KM & Prietula, MJ 1994, 'ACTS Theory: Extending the model of bounded rationality', *Computational Organisation Theory*, Lawrence Erlbaum Associates, Hillsdale, NJ.

Carley, KM 1999, 'On generating hypothesis using computer simulation', *Systems Engineering*, vol. 2, no. 2, pp. 69-77.

Chan, CL, Jiang JJ & Klein, G 2008 'Team Task Skills as a Facilitator for Application and Development Skills', *IEEE Transactions on Engineering Management*, vol.55, no. 3, pp. 434-441.

Checkland, P 1993, *Systems Thinking, Systems Practice*, John Wiley & Sons, England.

Choo, AS 2010, 'Impact of a Stretch Strategy on Knowledge Creation in Quality Improvement Projects', *IEEE Transactions on Engineering Management*, vol.58, no. 1, pp. 87-96.

Cooke, RA & Rousseau, DM 1983, 'The Factor Structure of Level I: Life Styles Inventory', *Educational and Psychological Measurement*, vol. 43, no. 2, pp. 449-457.

Cooke, RA & Szumal, JL 1994, 'The Impact of Group Interaction Styles on Problem-Solving Effectiveness', *The Journal of Applied Behavioral Science*, vol. 30, no. 4, pp. 415-437.

Cooke, RA & Szumal, JL 2000, 'Using the Organizational Culture Inventory to Understand the Operating Culture of Organisations', *Handbook of Organisational Culture & Climate*, Sage Publications, pp. 147-162, Thousand Oaks, CA.

DAF, Department of Air Force, 2000, *Guidelines for Successful Acquisition and Management*, version 3.0, May 2000, viewed 20 Nov. 2003,
<<http://web.nps.navy.mil/~menissen/mn3309/stsc-guidelines3.0/GSAMv3.pdf>>.

Dampney, K, Bush, P, Richards, D 2002, 'The Meaning of Tacit Knowledge', *Australasian Journal of Information Systems (AJIS)*, 2002, vol. 10, no. 1, pp. 2-13.

Darwin, C 1859, *On the Origin of Species by Means of Natural Selection*, John Murray, Albemarle Street, London.

DeMarco, T & Lister, T 1999, *Peopleware: Productivity, Projects and Teams*, Second Edition, 1999, Dorset House, New York, NY.

DeMarco, T 2009, 'Software Engineering: An Idea Whose Time Has Come and Gone?', *IEEE Software*, July/August 2009, vol. 26, no. 4, pp. 94-95.

Deming, WE 2000, 1994, *The New Economics: For Industry, Government, Education*, Second Edition 2000, MIT Press, Cambridge, MA.

Dijkstra, EW 1972, 'The Humble Programmer', 1972 ACM Turing Award Lecture, *Communications of the ACM*, vol. 15, no. 10, pp. 860-866.

DMO, 2006 'Project SEA 1411 ANZAC Ship Helicopter', Defence Materiel Organisation, viewed 12 June 2006,
<<http://www.defence.gov.au/dmo/asd/sea1411/sea1411.cfm>>.

DOD-AU, 2000, *Defence Annual Report 1999-2000*, Department of Defence (Australia), Commonwealth of Australia, Canberra, ACT.

DOD-AU, 2001, *Defence Annual Report 2000-2001*, Department of Defence (Australia), Commonwealth of Australia, Canberra, ACT.

DOD-AU, 2002, *Defence Annual Report 2001-2002*, Department of Defence (Australia), Commonwealth of Australia, Canberra, ACT.

DOD-AU, 2003, *Defence Annual Report 2002-2003*, Department of Defence (Australia), Commonwealth of Australia, Canberra, ACT.

DOD-AU, 2006, *Defence Capability Development Manual - DCDM 2006*, Department of Defence (Australia), Commonwealth of Australia, Canberra, ACT.

DOD-AU, 2008, ‘Seasprite Helicopters to be Cancelled’, *Department of Defence (Australia), Media Release*, 05 March 2008, MIN14/08, Commonwealth of Australia, Canberra, ACT.

DOD-AU, 2009a, *Defence Materiel Instruction (Procurement) - DMI (PROC)*, 13-0-001 V2.0, 2009, Department of Defence (Australia), Commonwealth of Australia, Canberra, ACT.

DOD-AU, 2009b, *Australian Standard Defence Contracting – ASDEFCON – Strategic Materiel*, V2.3, 2009, Department of Defence (Australia), Commonwealth of Australia, Canberra, ACT.

DOD-AU, 2010, *Defence Procurement Policy Manual, DPPM*, 1 October 2010, Department of Defence (Australia), Commonwealth of Australia, Canberra, ACT.

DOD-US, 2000, *Report of Defence Science Board Task Force on Defence Software*, November, 2000, Department of Defence (USA), Office of the Under Secretary of Defense For Acquisition and Technology Washington, DC.

EIA, 1999, *EIA-632 Standard: Processes for Engineering a System*, January 1999, Electronic Industries Alliance, Arlington, VA.

Fagiolo, G, Windrum P, & Moneta, A 2006, *Empirical validation of agent-based models: A critical survey*, (No. 2006/14), Sant’Ana School of Advanced Studies,

Laboratory of Economics and Management, Pisa, Italy, viewed 26 May 2007,
<<http://www.lem.sssup.it/WPLem/files/2006-14.pdf>>.

Eveleens, JL & Verhoef, C 2010, 'The Rise and Fall of the Chaos Report Figures', *IEEE Software*, January/February 2010, vol. 27, no. 1, pp. 30-36.

Ferguson, G & Blekin, M 2008, 'Seasprite: What went wrong?', *Australian Defence Magazine*, December 2008, Brisbane, QLD.

Fishman, C 1996, 'They Write the Right Stuff', *Fast Company*, Issue 06, December 1996, viewed 13 May 2006, <<http://www.fastcompany.com>>.

Florac, WA, Carleton, AD & Barnard, JR 2000, 'Statistical Process Control: Analysing a Space Shuttle Software Process', *IEEE Software*, July/August 2000, vol.17, no. 4, pp. 97-106.

Flowers, S 1996, *Software Failure: Management Failure*, John Wiley & Sons, West Sussex, UK.

Forester, JW 1958, 'Industrial Dynamics: A Major Breakthrough for Decision Makers', *Harvard Business Review*, vol. 36, no. 6, pp. 37-66.

GAO 2004, *Defence Acquisitions, Strong Management Practices Are Needed to Improve DOD's Software Intensive Weapon Acquisitions*, GAO-04-393, March 2004, United States General Accounting Office, Washington, DC.

GAO 2009, *Defence Acquisitions, Assessment of Selected Weapon Programs*, GAO-09-326SP, March 2009, United States General Accounting Office, Washington, DC.

Gell-Mann, M 1995, 'Complex Adaptive Systems', in *The Mind, The Brain and Complex Adaptive Systems*, pp.11-23, Addison-Wesley Publishing Company, Reading, MA.

Gilb, T 2005, *Competitive Engineering: A Handbook for Systems Engineering, Requirements Engineering, and Software Engineering Using PLANGUAGE*, Elsevier Butterworth-Heinemann, Burlington, MA.

Gilbert, N 2008, *Agent-Based Models*, SAGE Publications, Thousand Oaks, CA.

Glass, R 2005, 'IT Failure Rates – 70% or 10-15%', *IEEE Software*, May 2005, vol. 22, no. 1, pp. 110-112.

Glass, R 2006, 'The Standish Report: Does it Really Describes a Software Crises?', *Communications of the ACM*, vol. 49, no. 8, pp. 15-16.

Hitchins, DK 1992, *Putting Systems to Work*, John Willey & Sons, London, England.

Hitchins, DK 2003, *Advance Systems Thinking, Engineering and Management*, Artech House Inc., Norwood, MA.

Hill, R 2002, 'Defence Acquisition: getting it right', key note address *C3I Defence Watch Seminar*, 26 September 2002, Canberra, ACT, viewed 13 Nov. 2003, <<http://www.minister.defence.gov.au/HillSpeechtpl.cfm?CurrentId=1926>>.

Hill, R 2003, 'Seasprite helicopters showcased in Nowra', *Defence Ministers & Parliamentary Secretary Media Release (Australia)*, 18 Oct 2003, viewed 13 Nov. 2003, <<http://www.minister.defence.gov.au/Hilltpl.cfm?CurrentId=3208>>.

Hoffman, LR 1979, 'Applying Experimental Research on Group Problem Solving to Organisations', *The Journal Of Applied Behavioral Science*, July 1979, vol. 15, no. 3, pp. 375-391.

Holland, J 1995, *Hidden order: how adaptations builds complexity*, Addison-Wesley Publishing Company, USA.

Honour, E 2004, 'Understanding the Value of Systems Engineering', *2004 INCOSE International Symposium in Toulouse*, France, viewed 12 Feb. 2011, <<http://www.incose.org/secoe/0103/ValueSE-INCOSE04.pdf>>.

Humphrey, WS 1988, 'Characterizing the software process: a maturity framework', *IEEE Software*, March 1988, vol. 5, no. 2, pp. 73-7.

Humphrey, WS 2002, *Winning with Software: An Executive Strategy*, Addison-Wesley Publishing Company, USA.

Humphrey, WS 2010a, 'Why Can't We Manage Large Projects?', *CrossTalk The Journal of Defense Software Engineering*, July/August 2010, vol.23, no. 4, pp. 4-7.

Humphrey, WS 2010b, ‘An Interview With Watts S. Humphrey’, *CrossTalk The Journal of Defense Software Engineering*, July/August 2010, vol.23, no. 4, pp. 8-13.

INCOSE 2004, ‘What is Systems Engineering?’, *INCOSE Website*, updated 14 June 2004, viewed 15 Aug. 2011, <<http://www.incose.org/practice/whatissystemseng.aspx>>.

ISO/IEC STD-12207-2008, *IEEE STD-12207-2008, 2008, Systems and Software Engineering – Software Life Cycle Processes*, Second Edition, 2008-02-01, ISO/IEC/IEEE.

ISO/IEC STD-15288-2008, *IEEE STD-15288-2008, 2008, Systems and Software Engineering – System Life Cycle Processes*, Second Edition, 2008-02-01, ISO/IEC/IEEE.

Jones, C 1995, *Patterns of Software System Failure and Success*, International Thomson Computer Press, Boston, MA.

Jones, C 1996, *Applied Software Measurement – Assuring Productivity and Quality*, Second Edition, McGraw-Hill, USA.

Jones, C 2002, ‘Defense Software Development in Evolution’, *CrossTalk The Journal of Defense Software Engineering*, November 2002, vol. 15, no. 11, pp. 26-29.

Jones, C 2003, ‘Variations in Software Development Practices’, *IEEE Software*, Nov.-Dec. 2003, vol. 20, no. 6, pp. 22 – 27.

Jones, C 2006, ‘Social and Technical Reasons for Software Project Failure’, *CrossTalk The Journal of Defense Software Engineering*, June 2006, vol. 19, no. 6, pp. 4-9.

Jones, C 2008, *Applied Software Measurement – Global Analysis of Productivity and Quality*, Third Edition, McGraw-Hill, USA.

Johnson, NJ & Klee, T 2007, ‘Passive-Aggressive Behavior and Leadership Styles in Organizations’, *Journal of Leadership and Organizational Studies*, Nov. 2007, vol. 14, no. 2, pp. 130-142.

Jung, CG 1971, *Psychological Types*, Princeton, New Jersey, Princeton University Press, first published in German in 1921.

Kasser, JE 1998, ‘What Do You Mean, You Can’t Tell Me How Much of My Project Has Been Completed?’, *Proceedings of the First European Conference on Software Metrics (FESMA 98)*, Antwerp, Belgium, 1998, reprinted from FESMA 98, viewed 7 Aug. 2011,
[<http://www.therightrequirement.com/pubs/publications.htm>](http://www.therightrequirement.com/pubs/publications.htm).

Kauffman, S 1993, *The Origins of Order: Self-Organisation and Selection in Evolution*, Oxford University Press, New York, NY.

Kauffman, S 1995, *At home in the universe: the search for the laws of self-organisation and complexity*, Oxford University Press, New York, NY.

Keirsey, D & Bates, M 1984, *Please Understand Me: Character & Temperament Types*, Prometheus Nemesis Books, Ltd., Del Mar, CA.

Keirsey, D & Bates, M 1998, *Please Understand Me II: Temperament, Character, Intelligence*, Prometheus Nemesis Books, Ltd., Del Mar, CA.

Klein, M, Sayama, H, Faratin, P, Bar-Yam, Y 2001, ‘What complex systems research can teach us about collaborative design’, *Proceedings of the Sixth International Conference on Computer Supported Cooperative Work in Design (CSCWD-2001) 5-12 (IEEE Press 2001)*, viewed 10 Nov. 2005,
[<http://www.necsi.edu/research/engineering/>](http://www.necsi.edu/research/engineering/>).

Langton, C 1990, ‘Computation at the Edge of Chaos: Phase Transitions and Emergent Computation’, *Physica D: Nonlinear Phenomena*, vol. 42, issue 1-2, pp. 12-37.

Lawson, EC 2005, *Examination of social systems of engineering projects*, PhD dissertation, University of South Australia, Mawson Lakes, SA.

Leach, LP 2000, *Critical Chain Project Management*, Second Edition, Artech House Inc., Norwood, MA.

Linberg, KR 1999, ‘Software developer perceptions about software project failure: a case study’, *Journal of Systems and Software*, 1999, vol. 49, issue 2-3, pp. 177-192.

Maslow, AH 1943, 'A Theory of Human Motivation', *Psychological Review*, vol. 50, no. 4, pp. 370-396, viewed 7 Nov. 2009,
<http://www.ahpweb.org/articles/theory_human_motivation.html>.

McConnell, S 1996, *Rapid Development: Taming Wild Software Schedules*, Microsoft Press, Redmond, WA.

McDowell, J 2008, 'From the Source: Mr Jim McDowell, Managing Director BAE Systems Australia', interview, *Australian Defence Magazine*, 1 September 2008, viewed 5 March 2011, <<http://www.australiandefence.com.au/48A38B40-F807-11DD-8DFE0050568C22C9>>.

McDowell, J 2010, 'BAE boss plots a new course', *The Australian*, 23 October 2010, viewed 5 March 2011, <<http://www.theaustralian.com.au/national-affairs/defence/bae-boss-plots-a-new-course/story-e6frg8yo-1225940211644>>.

McKinnie, S 2003, 'DMO Reform: Implementing a Systems Philosophy in the Defence Materiel Organisation', *Systems Engineering, Test and Evaluation Conference 2003*, Canberra, ACT.

McLucas, AC 2001, *An investigation into the integration of qualitative and quantitative techniques for addressing systemic complexity into in the context of organisational strategic decision-making*, PhD thesis, Australian Defence Force Academy, Canberra ACT.

McIntosh, MK & Prescott, JB 1999, *Report for the Minister for Defence on the Collins Class Submarine and Related Matters*, Commonwealth of Australia 1999, Canberra, ACT.

Miller, JH & Page, SE 2007, *Complex Adaptive Systems: An Introduction to Computational Models of Social Life*, Princeton University Press, Princeton, NJ.

Moody, G 1998, 'The Wild Bunch', *New Scientist*, December 1998, no. 2164, p.42-46.

Newell, A, Yost, G, Laird, J, Rosenbloom, P & Altman, E 1990, 'Formulating the problem space computational model', *25th Anniversary Symposium, School of Computer Science*, Carnegie Mellon University, Pittsburgh, PA.

NOC, 2006, ‘The SH-2G(A) Kaman Super Seasprite’, *Naval Officers Club*, viewed 12 June 2006, <<http://www.navalofficer.com.au/seasprite.htm>>.

Nonaka, I 1994, ‘A Dynamic Theory of Organizational Knowledge Creation’, *Organization Science*, Feb. 1994, vol. 5, no. 1, pp. 14-37.

North, MJ, Macal, CM 2007, *Managing Business Complexity: Discovering Strategic Solutions with Agent-Based Modelling and Simulation*, Oxford University Press, New York, NY.

NRC, 1993, *An Assessment of Space Shuttle Flight Software Development Process*, National Research Council of Aeronautics and Space Engineering Board, The National Academy Press, Washington, D.C.

NRC, 2007, *Summary of a Workshop on Software-Intensive Systems and Uncertainty at Scale*, National Research Council of National Academies, The National Academy Press, Washington, DC.

Perkins, TK 2006, ‘Knowledge: The Core Problem of Project Failure’, *CrossTalk The Journal of Defense Software Engineering*, June 2006, vol. 19, no. 6, pp. 13-15.

Pfleeger, SL 1999, ‘Albert Einstein and empirical software engineering’, *Computer*, 1999, vol. 32, issue 10, pp. 32-38.

PMI, 2008, *A Guide to the Project Management Body Of Knowledge (Pmbok® Guide)*, Fourth Edition, Project Management Institute, Inc., Newtown Square, PA.

Popper, K 1959, *The Logic of Scientific Discovery*, Taylor & Francis e-Library, 2005.

Procaccino, JD, Verner, JM & Lorenzet, SJ, ‘Defining and Contributing to Software Development Success’, *Communications of the ACM*, August 2006, vol. 49, no. 8, pp.79-83.

Qinxuan, GU & Yingting, GU 2010, ‘Study on the Construct of Knowledge-Sharing Motivation’, *2010 International Conference on Manage and Service Sciences (MASS)*, Wuhan, China.

Rahmandad, H & Sterman, J 2008, 'Heterogeneity and Network Structure in the Dynamics of Diffusion: Comparing Agent-Based and Differential Equation Models', *Management Science*, May 2008, vol. 54, no. 5, pp. 998–1014.

Reber, AS & Reber, ES 2001, *Dictionary of Psychology*, Third Edition, Penguin Books, London.

Rechtin, E 2000, *Systems Architecting of Organisations: Why Eagles Can't Swim*, CRC Press, Boca Raton, FL.

Reynolds, WN & Dixon, D 2000, 'A General Framework for Representing Behaviour in Agent Based Modelling', *Complex Systems and Policy Analysis: New Tools for a New Millennium*, Arlington VA, September 2000, RAND Corporation Science & Technology Policy Institute, viewed 10 Nov. 2009,
<<http://www.leastsquares.com/papers/rand2000.pdf>>

Robbins, SP 2003, *Organisational Behavior*, Tenth Edition, Prentice Hall, Upper Saddle River, NJ.

Rogers, D 2008, 'The Electronic Warfare Self Protection Ground Systems of Project Echidna, AIR 5416, Phase 2A', *7th International Symposium and Exhibition – Australian Chapter – Association of Old Crows*, 26-27 May 2008, Adelaide, SA.

Ropohl, G 1999, 'Philosophy of Socio-Technical Systems', *Society for Philosophy and Technology*, Spring 1999, vol. 4, no. 3, Electronic Journals, viewed 12 Oct. 2008,
<http://scholar.lib.vt.edu/ejournals/SPT/v4_n3html/ROPOHL.html>.

Rulke, DL & Galaskiewicz, J 2000, 'Distribution of Knowledge, Group Network Structure, and Group Performance', *Management Science*, May 2000, vol. 46, no. 5, pp. 612-625.

Ryan, RM & Deci, EL 2000, 'Intrinsic and Extrinsic Motivations: Classic Definition and New Directions', *Contemporary Educational Psychology*, January 2000, vol. 25, no. 1, pp. 54-67.

Saaty, TL 2000, *Fundamental of Decision Making and Priority Theory with the Analytic Hierarchy Process*, vol. VI of AHP Series, RWS Publications, Pittsburgh, PA.

Schmidt, B 2000, *The Modelling of Human Behaviour*, Gruner Druck GmbH, Erlangen, Germany.

Scott, R, Holdanowicz , G & Bostock, I 2004, 'Second coming for the Super Seasprite', *The Aviation Forum*, viewed 13 June 2006,
<<http://forum.keypublishing.co.uk/showthread.php?t=24443>>.

SEI, 2001, People *Capability Maturity Model, P-CMM, V2.0*, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA.

SEI, 2002, *Software Acquisition Capability Maturity Model, SA-CMM, VI.03*, March, 2002, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA.

SEI, 2006, *Ultra-Large-Scale Systems: The Software Challenge of the Future*, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA.

SEI, 2010, 'Obituary: National Medal of Technology Winner Watts Humphrey, 1927 – 2010', Software Engineering Institute, 28 October 2010, viewed 10 March 2011,
<http://www.sei.cmu.edu/newsitems/Humphrey_obituary.cfm>.

Senge, PM 2006, *The Fifth Discipline: The Art & Practice of the Learning Organisation*, Second Edition, Random House Australia, Sydney, NSW.

Silverman, BG, Cornwell, J, Johns, M & O'Brien, K 2004, 'Human Behavior Models for Agents in Simulators and Games: Part I: Enabling Science with PMFserv', *13th Conference on Behavior Representation in Modelling and Simulation (BRIMS)*, 17-20 May 2004, Arlington, VA.

Stacey, RD 1999, *Complexity and Creativity in Organisations*, Berrett-Koehler Publishers, San Francisco, CA.

Standish Group, 1994, 'The CHAOS Report', Standish Group International, Inc., viewed 13 March 2004,
<http://www.standishgroup.com/sample_research/PDFpages/chaos1994.pdf>.

Standish Group, 1995, 'Unfinished Voyages: A Follow-Up To The CHAOS Report', Standish Group International, Inc., viewed 7 August 2011,
<http://www.standishgroup.com/sample_research/index.php>.

Standish Group, 1999, ‘CHAOS: A receipt for success’, Standish Group International, Inc., viewed 7 August 2011,
<http://www.standishgroup.com/sample_research/index.php>.

Standish Group, 2001, ‘Extreme CHAOS’, Standish Group International, Inc., viewed 13 Nov. 2003,
<http://www.standishgroup.com/sample_research/PDFpages/ExtremeChaos2001.pdf>.

Standish Group, 2003, ‘Latest Standish Group CHAOS Report shows project success rates have improved by 50%’, Press Release, Standish Group International, Inc., viewed 13 Nov. 2003 <<http://www.standishgroup.com/press/>>.

Standish Group, 2005, ‘CHAOS Rising: A CHAOS Executive Commentary’, Standish Group International, Inc., viewed 7 August 2011,
<http://www.standishgroup.com/sample_research/index.php>.

Standish Group, 2009, ‘CHAOS Summary 2009: The 10 Laws of CHAOS’, Standish Group International, Inc., viewed 7 August 2011,
<<http://www.education.state.pa.us/>>.

Sterman, JD 2002, ‘All models are wrong: reflections on becoming a systems scientist’, *System Dynamics Review*, vol. 18, no. 4, Winter 2002, pp. 501–531, viewed 13 Jan. 2010, <http://jsterman.scripts.mit.edu/On-Line_Publications.html>.

Stevens, R, Brook P, Jackson K & Arnold, S 1998, *Systems Engineering: Coping with Complexity*, Prentice Hall, London.

Te’eni, D 2001, ‘A Cognitive-Affective Model of Organizational Communication for Designing IT’, *MIS Quarterly*, vol. 25, no. 2, June 2001, pp. 251-312.

Thite, M 1999, ‘Leadership: A Critical Success Factor in IT Project Management’, *Portland International Conference on Management of Engineering and Technology*, 1999, PICMET '99, 25-29 July 1999, vol.2, pp. 298-303, Portland, OR.

- Thomson, M 2011, *The Cost of Defence – ASPI Defence Budget Brief 2011-2012*, Australian Strategic Police Institute, viewed 23 May 2011, <<http://www.aspi.org.au/publications/>>.
- Turner, R & Boehm, B 2003, ‘People Factors in Software Management: Lessons From Comparing Agile and Plan-Driven Methods’, *CrossTalk The Journal of Defense Software Engineering*, December 2003, vol. 16, no. 12, pp. 4-8.
- Turner, R & Müller, R 2005, ‘The Project Manager’s Leadership Style as a Success Factor on Projects: A Literature Review’, *Project Management Journal*, June 2005, pp.49-61.
- Turner, R 2007, ‘Towards Agile Systems Engineering Processes’, *CrossTalk The Journal of Defence Software Engineering*, April 2007, vol. 20, no 4, pp 11-15.
- Unknown Reader, 1998, ‘The True Costs of Linux Development’, *The Register*, 21 October 1998, viewed 15 May 2007, <http://www.theregister.co.uk/1998/10/21/the_true_costs_of_linux/>.
- USN, United States Navy 2008, *Guidebook for Acquisition of Naval Software Intensive Systems*, version 1.0, September 2008, Office of the Assistant Secretary of the Navy (Research, Development and Acquisition), viewed 12 May 2011, <http://acquisition.navy.mil/organizations/dasns/rda_cheng>.
- Vose, D 2000, *Risk Analysis, A Quantitative Guide*, Second Edition, John Wiley & Sons, Ltd, Chichester, England.
- Wheeler, DA 2006, ‘Linux Kernel 2.6: It's Worth More!’, viewed 5 May 2007 <<http://www.dwheeler.com/essays/linux-kernel-cost.html>>.
- Wiener, N 1948, *Cybernetics or Control and Communication in the Animal and the Machine*, John Wiley & Sons Inc., New York, NY.
- Yang, LR & Chen, YT 2010, ‘Project Manager’s Leadership Style Lnked with Schedule and Cost Performance’, *International Conference on Management and Service Science (MASS)*, 2010, 24-26 August 2010, Wuhan, China.
- Yeh, YJ & Chou, HW 2005, ‘Team Composition and Learning Behaviours in Cross-Functional Teams’, *Social Behaviour and Personality*, vol. 33, no. 4, pp. 391-402.

Yost, G & Newell, A 1989, 'A problem space approach to expert systems specification', in *Proceedings of the 11th International Joint Conference on Artificial Intelligence*, vol. 1, pp. 621-62,. San Mateo, CA.

Yule, P & Wolner, D 2008, *The Collins Class Submarine Story: Steel, Spies and Spin*, Cambridge University Press, Melbourne, VIC.

Xiao-qing, B & Zhou, N 2010, 'Research on Inspiration Mechanism of Knowledge Sharing Based Motivations', *2010 International Conference on E-Business and E-Government (ICEE)*, pp.1883-1886, Guangzhou, China.

Appendix A: Simulation Environment

A.1. Overview

The Software-intensive Acquisition Agent Based Model Simulation (SIAABMSim) developed for this research is a *Repast 3.1*⁴⁶ Java application developed in the *eclipse*⁴⁷ software development environment (SDE). SIAABMSim implements the model presented in Chapter 5.

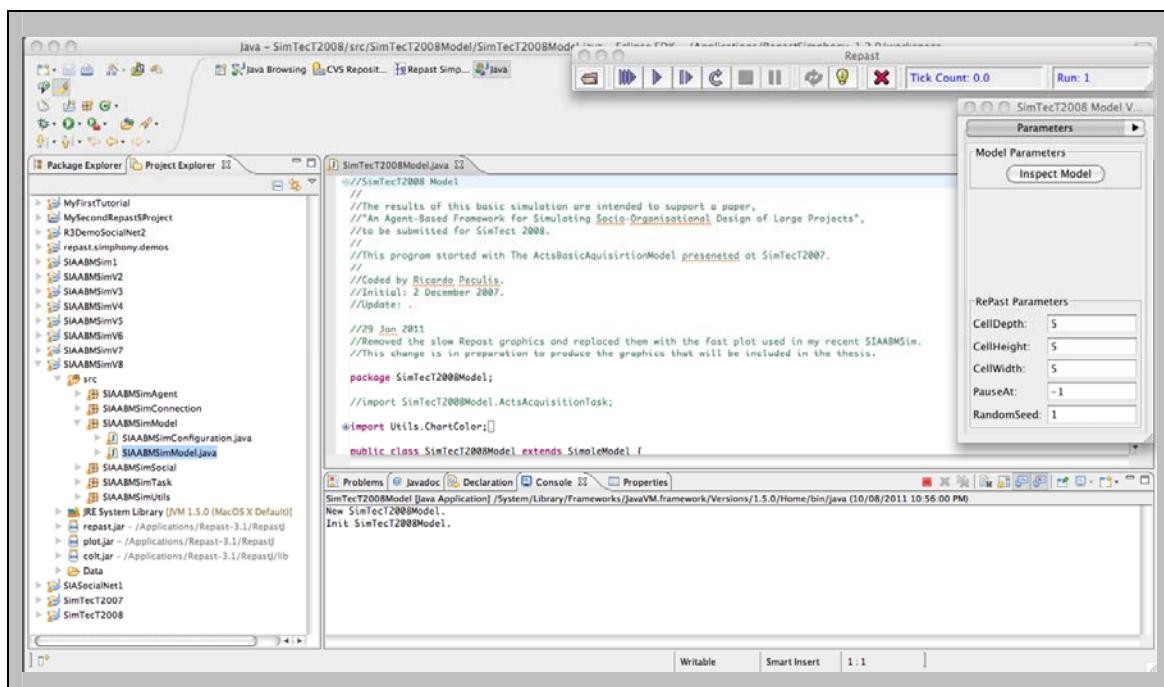


Figure A-1 – SIAABMSim SDE

SIAABMSim accepts configuration files that specify scenarios defining the task and the social system and parameters that influence many aspects of the simulation. Agents can be *actors* and *artefacts*. Actors are cognitive agents that represent people and artefacts are non-cognitive agents that are the products created and modified by actors. Configuration files specify artefacts, dependencies between artefacts, actors and how they interact with other actors and with artefacts.

⁴⁶ Recursive Porous Agent Simulation Toolkit (Repast) is a Java based toolkit for agent-based modelling and simulation available from http://repast.sourceforge.net/repast_3.

⁴⁷ **eclipse** is a software development environment available from <http://www.eclipse.org/>.

A.2. Displays

The screenshot below shows the three main views of the simulation. The larger window is the *Environment Window* and the two graphic windows are the *Behaviour Window* and the *Performance Window*.

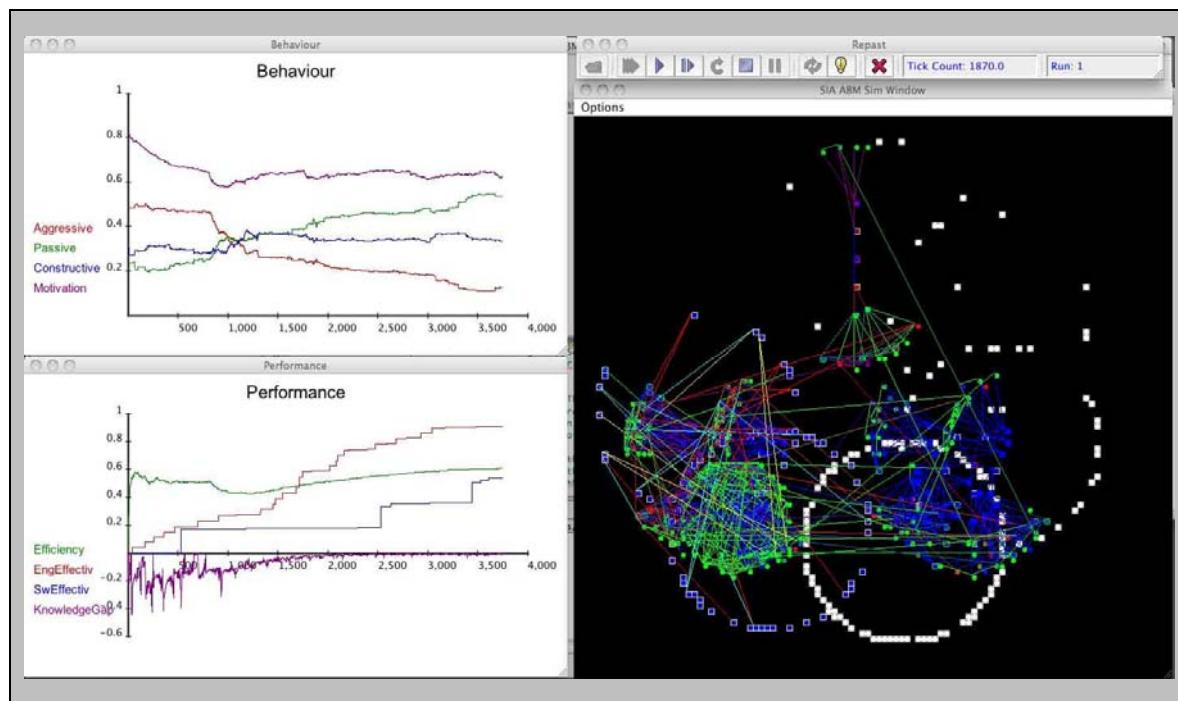


Figure A-2 – SIAABMSim Main Screen

Environment Window

The Environment Window shows actors, artefacts, tasks, interactions and dependencies. The display has five views that can be selected to display Artefacts, Dependencies, Actors, Interactions and Tasks. The display below has the Dependencies view switched off.

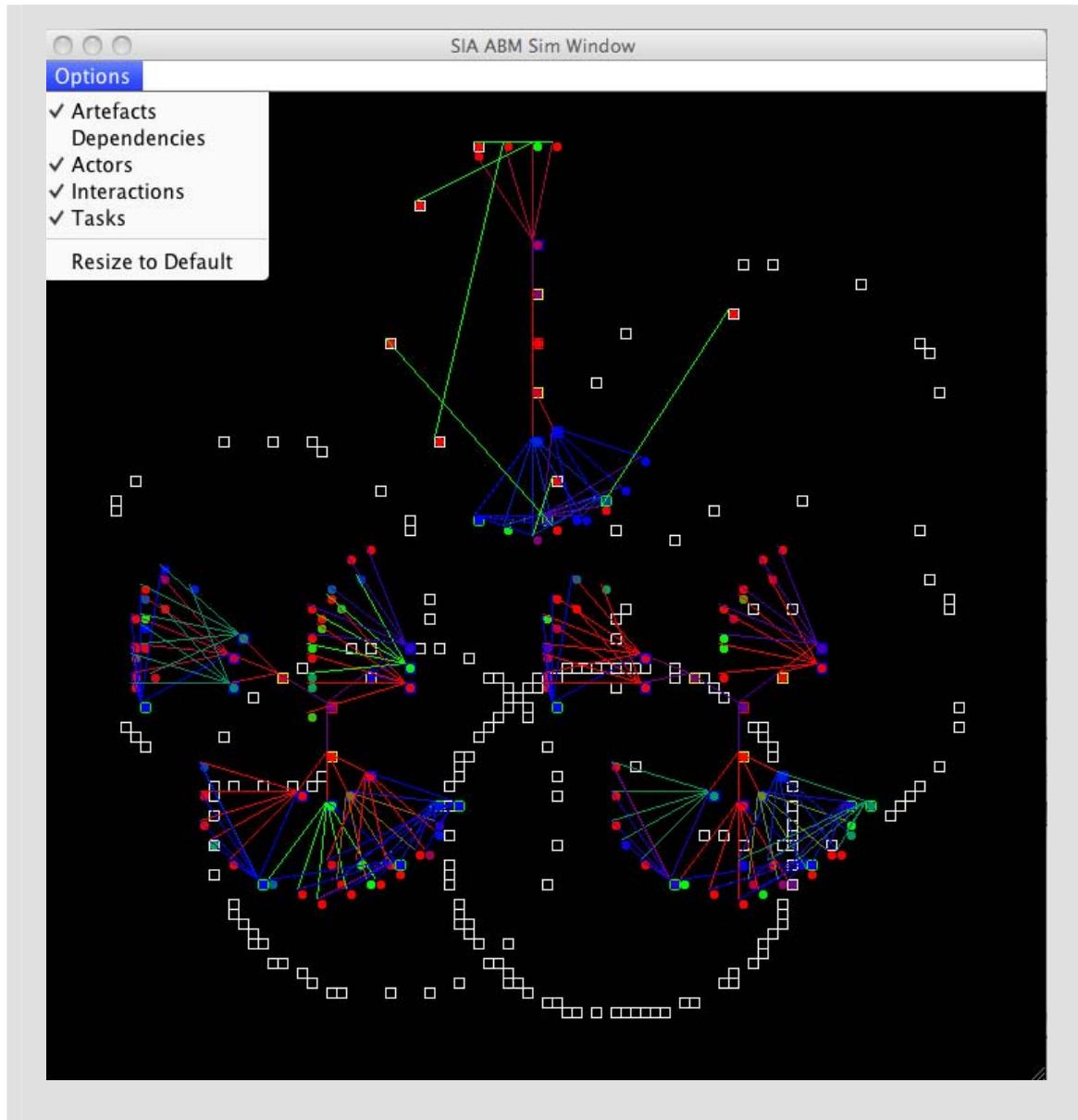


Figure A-3 – Environment Window

Products and Artefacts

Artefacts are represented by white squares colour filled. The fill colour indicates the effectiveness of the artefact as the task progresses, as shown below. If effectiveness is zero (black), the artefact has been planned but the development has not started.

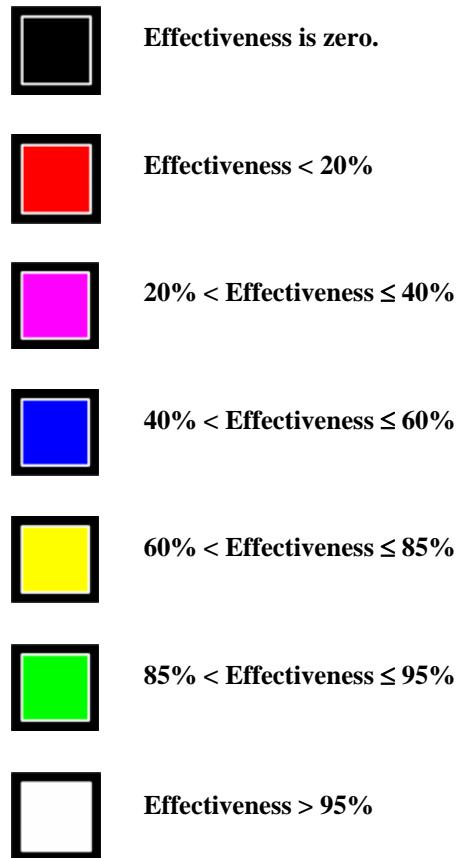


Figure A-4 – Artefact Effectiveness Colour Code

Artefacts that belong to the same product are grouped in circles. Parallel development can occur in accordance with the development life cycle model but for convenience a nominal output order is assumed and placed in a circle in a clockwise manner. Starting from the top the products are: Statement of Need; Capability Development; System Specification; System Architecture; Software Specification and Design; and the Software Product, should satisfy the Need, as shown in the screenshot below. The overlap of circles has no practical meaning.

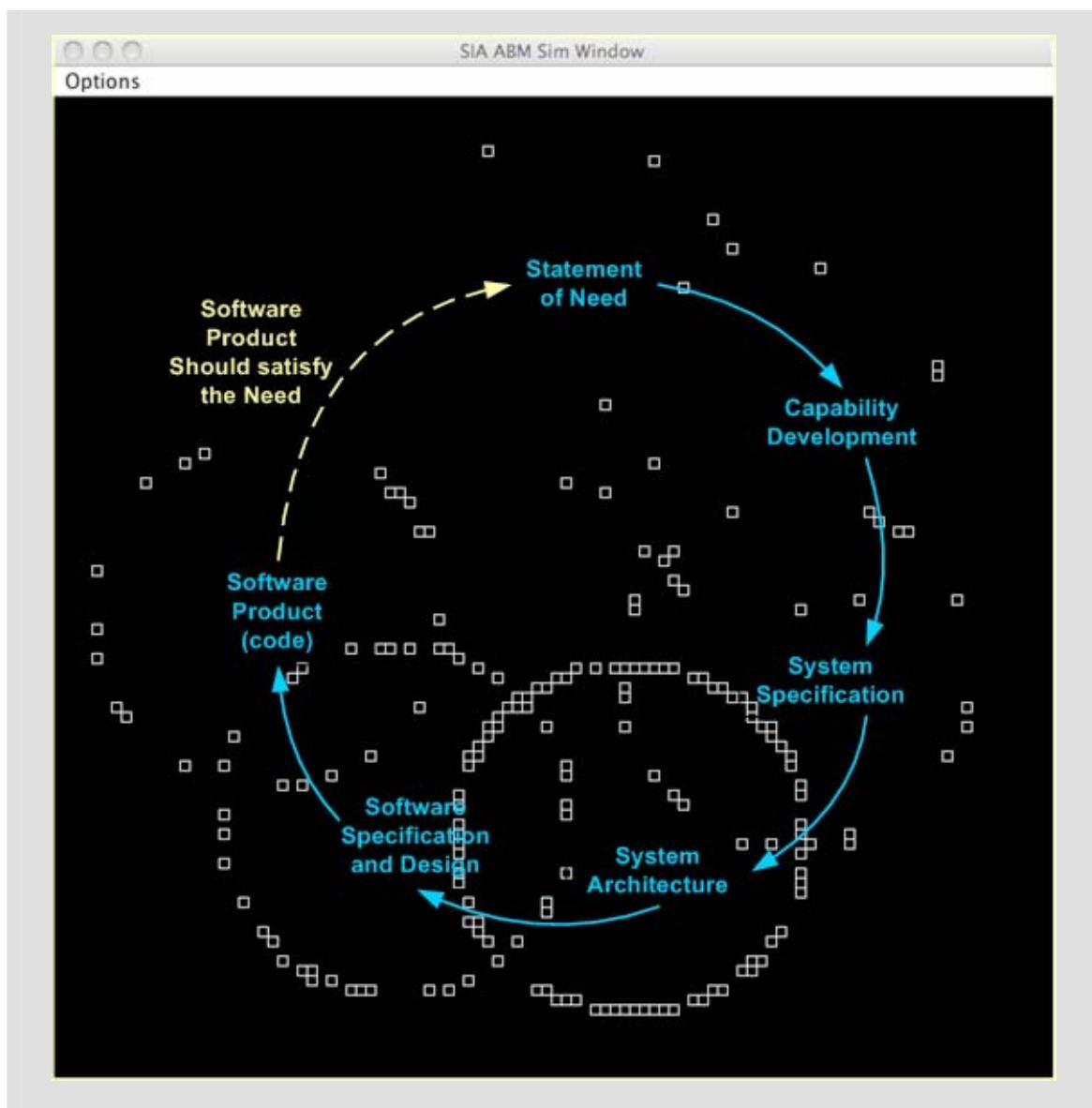


Figure A-5 – Environment Window – Artefacts View

Dependencies are shown as white lines connecting artefacts. Dependencies correspond to connections between artefacts through coefficients of Transformation Matrices, i.e. the output artefact depends on a transformation of an input artefact. As dependencies do not change dynamically the Dependencies layer is usually switched off.

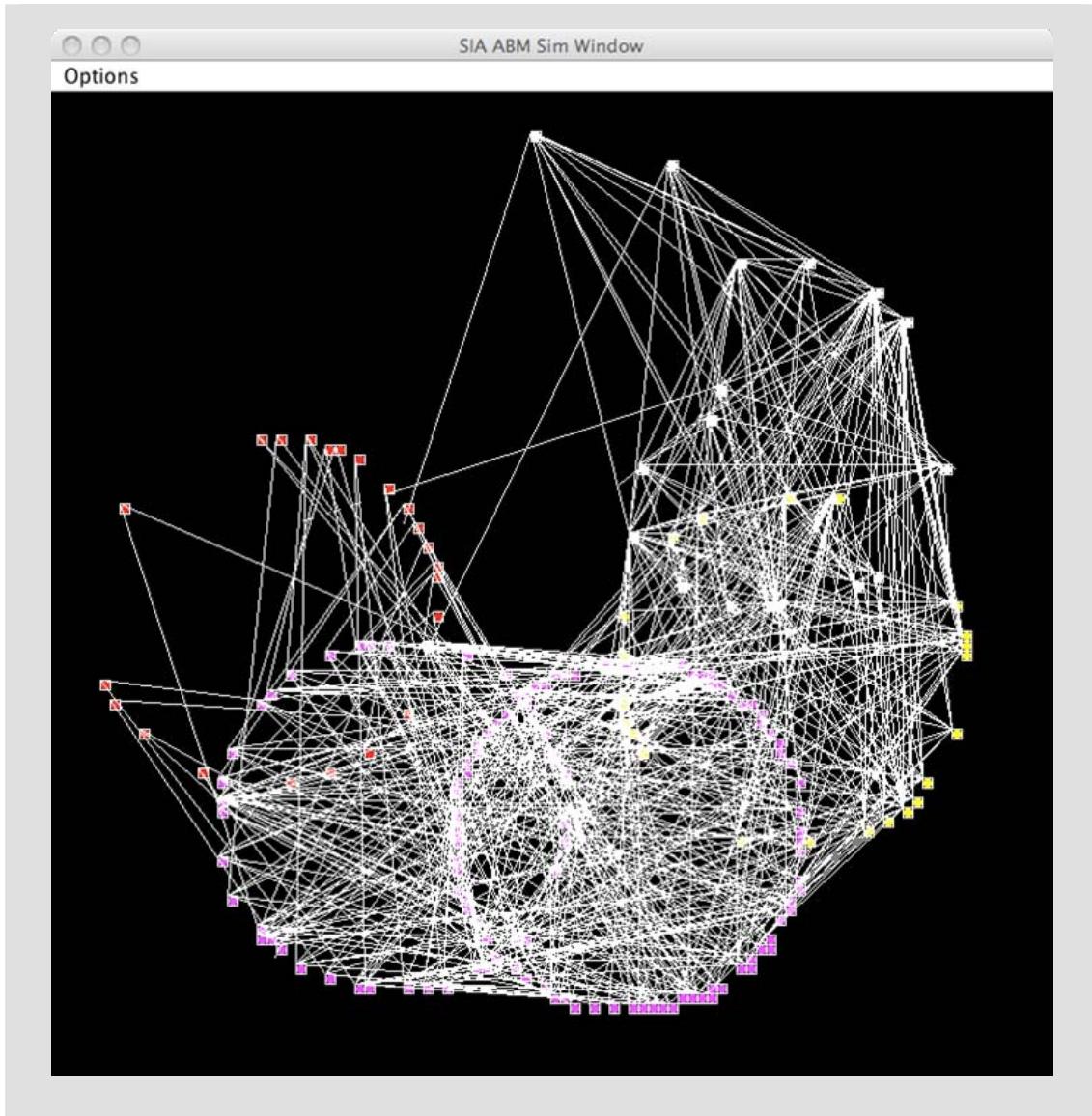


Figure A-6 – Environment Window – Artefacts and Dependencies Views

Actors, Interactions, Teams and Organisations

Actors are shown as a symbol that reflects the actor's role and the actor's behavioural style. Team Members (TM) are represented by a coloured circle and the fill colour indicates the actor's behavioural style. Aggressive style is red; constructive is blue; and passive is green. As behaviour changes the colour hue assumes values between red, blue and green. The symbol for each of the actor's roles is shown below.

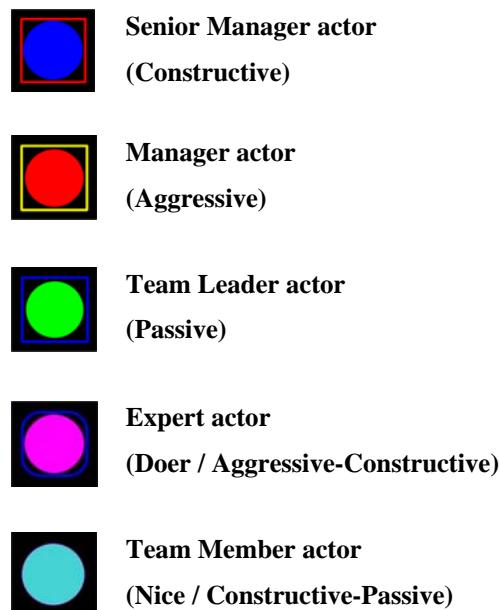


Figure A-7 – Actor Role Symbols

Interaction is represented by a coloured line connecting the two actors. The behaviour of the actor that originates the interaction determines the colour of the line. Formal interactions create the structure of teams, departments and organisations. Informal interactions may connect actors from different teams and organisations.

Organisations have one Senior Manager and one or many Managers. Departments have one Manager and one or many Team Leaders. Teams have one Team Leader and one or many Team Members. A configuration file defines the structure of the organisation. The screenshot that follows shows three organisations. The organisation at the top of the window has the Senior Manager, two Managers, three Team Leaders, three Experts and

twelve Team Members. The way that the actors and interactions are placed on the display represents the structure of the organisation. Organisations are placed close to the artefacts that are their responsibility.

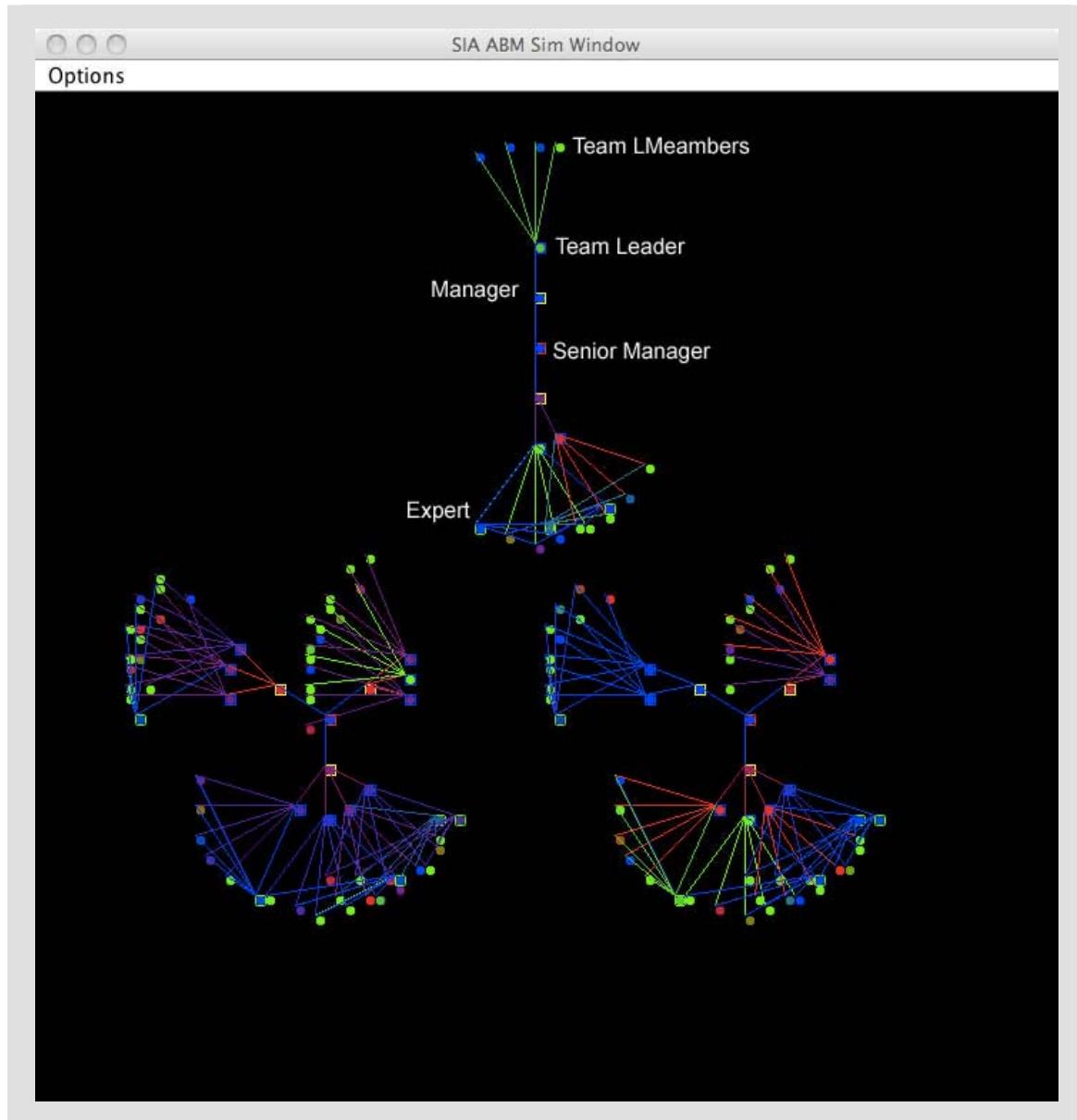


Figure A-8 – Environment Window – Actors and Interactions Views

The colour of actors and interactions changes as the simulation progresses reflecting the dynamic behaviour of the organisations.

Behaviour Window

The Behaviour Window shows the collective dynamic behaviour of the actors over time. The graphic shows the average of motivation and the three components (aggressive, constructive and passive) of behavioural style of the actors.

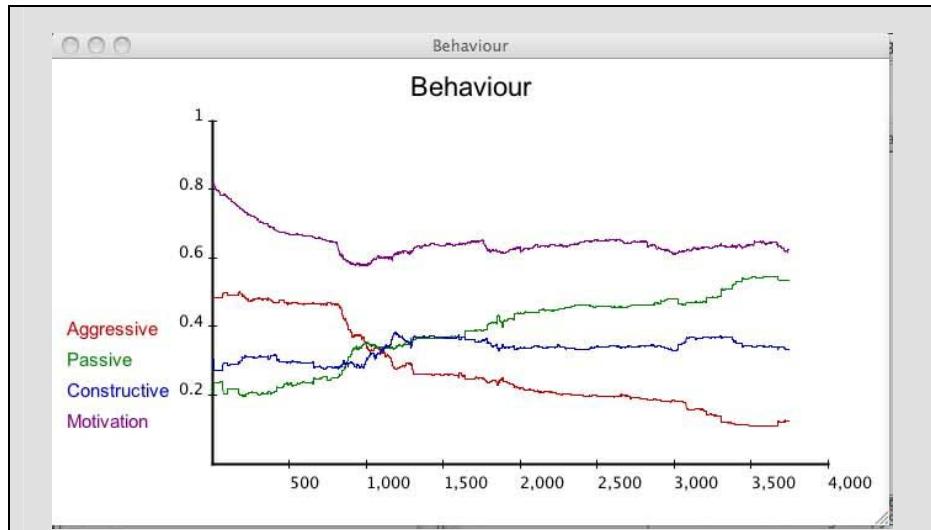


Figure A-9 – Behaviour Window

Performance Window

The Performance Window shows the progress of the task and the knowledge gap between what the tasks require and what is available in the population of actors.

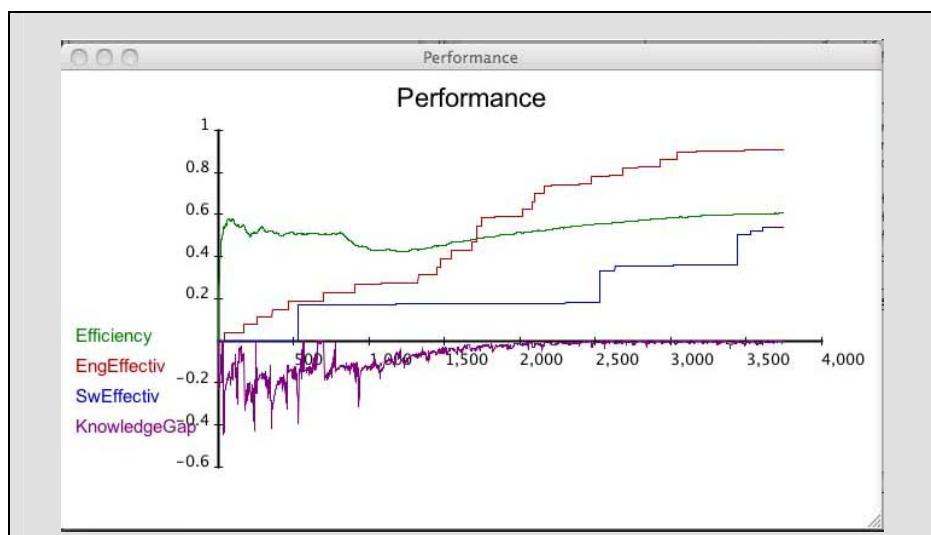


Figure A-10 – Performance Window

The attributes displayed in the Performance Window are normalised. Ideal effectiveness is represented by the numeric value 1.0, meaning that all the artefacts achieved what was intended and needed. Software Effectiveness (SwEffectiv) shown in blue is the effectiveness of the final product delivered to the customer. If the effectiveness of the software product does not achieve the ideal value, i.e. it is less than 1.0, the solution will not satisfy the need. Engineering Effectiveness (EngEffectiv), shown in red, is the effectiveness of all engineering artefacts that are needed to develop the software. If Engineering Effectiveness does not achieve the ideal value the software product will not achieve maximum effectiveness.

The Knowledge Gap showing negative in magenta indicates that the knowledge available in the population of actors is not sufficient to execute the task. When the actors collaborate and learn, the knowledge gap decreases as shown on the graphic. The high frequency noise-like variation in the knowledge gap is caused because every task has its own knowledge requirement and every actor has their own knowledge capacity. The high variation of knowledge required and knowledge available cause a high variation in the knowledge gap.

Efficiency is the ratio of effort spent on productive tasks and the total effort spent. Efficiency is lower when the actors are spending time collaborating and learning or reviews are in progress.

A.3. Simulation Parameters

Three files shown on the screenshot below control the simulation: Organisations File defines actors, teams and organisations; Probabilities File determines the probability of an actor's actions; and SimSettings File contains parameters that influence the whole simulation, including the development life cycle model, rates and limits.

The Actors File can be used to define actors in the population individually. The actors created from a distribution defined in the Organisations File can be exported, modified and imported by the simulation. ‘KeepPopulation’ and ‘MultiRunID’ are used when the simulation is executed automatically multiple times and in that case the results are recorded in a Simulation Results File identified by the ‘MultiRunID’. While in Multi-Run the simulation can use the same population of actors for all runs or generate a new population for each run.

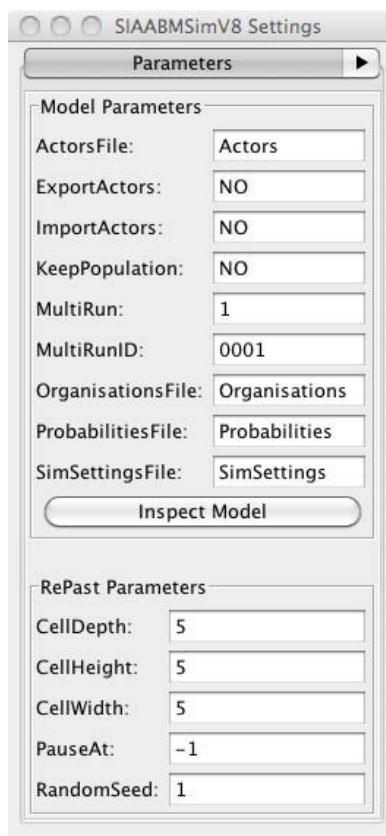


Figure A-11 – Simulation Parameters

population of actors for all runs or generate a new population for each run.

RePast Parameters are provided as the default. CellDepth, CellHeight and CellWidth are not used by SIAABMSim. PauseAt can be used to pause the simulation at a specified time count and a negative value is used when pause is not needed. RandomSeed is used for the pseudo-random number generator in RePast.

Organisations File

The Organisations File defines the population of actors and how the actors are organised in teams, departments and organisations. Below is a sample of the Organisations File defining three organisations.

Organisations.txt								
ORG	ORG1	0.2	Nil	Application				
CEO	1	0.5	0.4	1.0	0.0	1.0	0.0	1.0
Manager	4	0.5	0.4	1.0	0.0	1.0	0.0	1.0
Expert	8	0.33	0.0	1.0	0.0	1.0	0.0	1.0
TeamLeader	8	0.5	0.4	0.9	0.05	0.8	0.05	0.8
TeamMember	36	0.5	0.4	0.8	0.05	0.7	0.05	0.8
ORG	ORG2	0.2	Capabilities	SysEng	SysArchitect			
CEO	1	0.5	0.4	1.0	0.0	1.0	0.0	1.0
Manager	18	0.5	0.4	1.0	0.0	1.0	0.0	1.0
Expert	48	0.33	0.0	1.0	0.0	1.0	0.0	1.0
TeamLeader	48	0.5	0.4	0.9	0.05	0.8	0.05	0.8
TeamMember	240	0.5	0.4	0.8	0.05	0.7	0.05	0.8
ORG	ORG3	0.2	SoftEng	SoftDev				
CEO	1	0.5	0.4	1.0	0.0	1.0	0.0	1.0
Manager	24	0.5	0.4	1.0	0.0	1.0	0.0	1.0
Expert	66	0.33	0.0	1.0	0.0	1.0	0.0	1.0
TeamLeader	66	0.5	0.4	0.9	0.05	0.8	0.05	1.0
TeamMember	324	0.5	0.4	0.8	0.05	0.7	0.05	0.8

Figure A-12 – Organisations File

The file can specify one or many organisations. Each organisation starts with a label ORG followed by the name of the organisation (ORG1). The numeric value that follows is the probability of interactions between actors of different teams and departments. If the number is zero there will be no interactions outside the team. If the value is 1.0 the actors will interact with all the actors in the organisation, while a value of 0.20 shows a 20% chance of interactions outside the team to occur.

The words that follow the probability of interactions indicate the type of work that the organisation will do. The accepted words are Nil, Application, Capabilities, SysEng, SysArchitect, SwEng and SwDev. Nil represents the need and is the first input to the acquisition. The other words represent the ‘Statement of Need’ (Application), ‘Capability Development’ (Capabilities), ‘System Specification’ (SysEng), ‘System Architecture’ (SysArchitect), ‘Software Specification and Design’ (SwEng) and the ‘Software Product’ (SwDev). If there is only one organisation all the six products should be allocated to the organisation. When there are six organisations one single product should be allocated to each organisation.

The five lines following the organisation's attributes specify the population of actors in accordance with the format shown below.

ROLE	N	B	BSTD	K	KSTD	E	ESTD	M	MSTD
CEO	1	0.1	0.2	0.9	0.1	0.9	0.1	0.9	0.1
Manager	2	0.1	0.2	0.9	0.1	0.9	0.1	0.9	0.1
Expert	3	0.33	0.1	0.9	0.1	0.9	0.1	0.9	0.1
TeamLeader	3	0.33	0.5	0.8	0.2	0.7	0.2	0.9	0.1
TeamMember	12	0.33	0.5	0.7	0.3	0.6	0.3	0.8	0.2

Attribute	Description
ROLE	Role to which the population applies.
N	Number of actors in the population
B	Average <i>behavioural style</i> of the actors in the population: 0.0 = Aggressive; 0.33 = Constructive; 0.66 = Passive
BSTD	Standard Deviation of the <i>behavioural style</i> of the actors in the population.
K	Average <i>knowledge</i> of the actors in the population. 0.0 = no knowledge; 1.0 = perfect knowledge
KSTD	Standard Deviation of the <i>knowledge</i> of the actors in the population.
E	Average <i>experience</i> of the actors in the population. 0.0 = no experience; 1.0 = ideal experience
ESTD	Standard Deviation of the <i>experience</i> of the actors in the population.
M	Average <i>motivation</i> of the actors in the population. 0.0 = not motivated; 1.0 = totally motivated
MSTD	Standard Deviation of the <i>motivation</i> of the actors in the population.

Figure A-13 – Organisations File Format

Probabilities File

The Probabilities File defines the probability of events to occur in accordance with the actor's behavioural style. Events are actions performed by the actor or changes in its behaviour. The data is organised in a table where rows are the events and the columns the behavioural style. Below is a sample of the Probabilities File used to test the first and second hypotheses.

Header	Aggressive	Doer	Constructive	Nice	Passive	Neutral
Start	0.05	0.10	0.20	0.10	0.05	0.01
Respond	0.50	0.80	0.90	0.80	0.70	0.50
AskHelp	0.05	0.20	0.50	0.30	0.10	0.05
OfferHelp	0.02	0.05	0.80	0.10	0.05	0.01
AcceptHelp	0.30	0.40	0.80	0.50	0.20	0.10
Help	0.05	0.10	0.50	0.30	0.10	0.05
Praise	0.01	0.05	0.30	0.20	0.05	0.0
Reprimand	0.50	0.30	0.05	0.01	0.0	0.0
Learning	0.10	0.20	0.30	0.10	0.05	0.01
Forgetting	0.05	0.10	0.15	0.20	0.25	0.30
Motivated	0.10	0.15	0.20	0.15	0.10	0.15
Demotivated	0.20	0.25	0.40	0.25	0.10	0.15
Working	0.90	0.98	0.95	0.90	0.80	0.85
Overtime	0.70	0.50	0.30	0.10	0.05	0.10
Aggressive	0.50	0.40	0.30	0.20	0.1	0.0
Constructive	0.10	0.20	0.80	0.30	0.10	0.0
Passive	0.03	0.05	0.10	0.15	0.25	0.20
BAggressive	0.0	0.0	0.0	0.0	0.0	0.0
BConstructive	0.0	0.0	0.0	0.0	0.0	0.0
BPassive	0.0	0.0	0.0	0.0	0.0	0.0
MngAggressive	0.0	0.0	0.0	0.0	0.0	0.0
MngConstructive	0.0	0.0	0.0	0.0	0.0	0.0
MngPassive	0.0	0.0	0.0	0.0	0.0	0.0
TLAggressive	0.50	0.40	0.30	0.20	0.1	0.0
TLConstructive	0.20	0.20	0.5	0.30	0.10	0.0
TLPassive	0.03	0.05	0.10	0.15	0.25	0.20
QualityMotiv	0.2	0.3	0.7	0.6	0.55	0.5
CostSchedMotiv	0.8	0.7	0.3	0.4	0.35	0.5
ExplicitReward	0.7	0.6	0.3	0.4	0.55	0.5
ImplicitReward	0.3	0.4	0.7	0.6	0.35	0.5

Figure A-14 – Probabilities File

Table A-1 describes the events in the Probabilities File.

Table A-1 – Events in the Probabilities File

Event	Description
Start	Starts interaction with other actor
Respond	Responds interaction started by other actor
AskHelp	Asks help
OfferHelp	Offers help
AcceptHelp	Accepts help
Help	Helps actor when asked
Praise	Praises actor when conditions arise
Reprimand	Reprimands an actor when conditions arise
Learning	Spends productive time learning
Forgetting	Decreases knowledge and experience
Motivated	Increases motivation
Demotivated	Decreases motivation
Working	Works on productive task
Overtime	Works overtime on productive task, learning or helping other actors
Aggressive	Increases aggressive behaviour when conditions arise
Constructive	Increases constructive behaviour when conditions arise
Passive	Increases passive behaviour when conditions arise
MngAggressive	Manager actor increases aggressive behaviour when conditions arise
MngConstructive	Manager actor increases constructive behaviour when conditions arise
MngPassive	Manager actor increases passive behaviour when conditions arise
TLAggressive	Team Leader actor increases aggressive behaviour when conditions arise
TLConstructive	Team Leader actor increases constructive behaviour when conditions arise
TLPassive	Team Leader actor increases passive behaviour when conditions arise
BonusReward	Likes bonus as a form of reward
RecognitionReward	Likes recognition as a form of reward

SimSettings File

The simulation settings described in Table A-2 are used to set conditions and tune the simulation.

Table A-2 – SimSettings File Description

Parameter	Value	Description
GraphicDisplay	Y	Turns the displays on or off.
Development	E	Development life cycle: S = Sequential; E = Evolutionary The development life cycle comprises of a sequence of development phases each performed in one or many increments. Sequential Development mimics the waterfall life cycle where the end of a development increment enables the next increment of this and the next phase. Evolutionary Development forces a complete development cycle to be complete to enable the start of another increment. This option mimics the evolutionary life cycle where a partial but complete product has to be delivered to enable the next evolutionary cycle can start.
IterativeFactor	0.3	Proportion of what is learned by doing the task that is feedback to previous transformations. It can be seen as ‘feedback learning factor’
nTaskIncrements	4	Number of development increments
TargetEffectiveness	1.0	Target effectiveness
TargetEffort	-1	Target effort: -1 = no limit
TargetDuration	-1	Target duration: -1 = no limit
ExplicitReward	0.0	Percentage of explicit reward
ImplicitReward	0.0	Percentage of implicit reward
PraiseThreshold	1.1	Threshold for managers praise their subordinates
ReprimandThreshold	1.35	Threshold for managers reprimand their subordinates
DelayPraise	1000	Number of units of time before managers start praising
DelayReprimand	1000	Number of units of time before managers start reprimanding
MinKEM	0.4	Minimum value for Knowledge, Experience and Motivation
EffortFactor	1.0	Multiplier factor for task allocated effort
ReworkEffortFactor	1.0	Multiplier factor for rework allocated effort
ReviewFactor	1.0	Proportion of required Review effort that is assigned to the task
ReworkFactor	1.0	Proportion of required Work effort that is assigned to the task
TaskProgressRate	1.0	Coefficient used to calibrate the progress of tasks
LearningRate	0.3	Coefficient used to calibrate the process of learning
ForgettingRate	0.02	Coefficient used to calibrate the process of forgetting
ExperienceRate	0.3	Coefficient used to calibrate the process of acquiring experience
BehaviourRate	0.01	Coefficient used to calibrate the process of changing behaviour
ConstructiveRate	0.05	Coefficient used to calibrate the process of becoming constructive
PassiveRate	0.15	Coefficient used to calibrate the process of becoming passive
AggressiveRate	0.10	Coefficient used to calibrate the process of becoming aggressive
MotivationRate	0.05	Coefficient used to calibrate the process of motivation
RemoteInteractions	0.1	Probability of an interaction outside the team to occur
InteractionFactor	0.8	Probability of and interaction to occur
LearningFactor	0.4	Coefficient used to calibrate the process of learning by individual effort
CollaborationFactor	0.8	Coefficient used to calibrate the process of learning by collaboration
IterativeLearningFactor	1.0	Coefficient used to calibrate the process of learning by evolutionary development
FlipBehaviour	0.0	Probability for an actor randomly change behaviour
CopySenior Behaviour	0.0	Probability of Senior Managers and Managers to follow the behaviour of their customers or bosses

Transformations File

The Transformations File, shown below, specifies the name of the tasks and the transformation matrices files. The two numbers on each row are respectively the mean value of knowledge required to perform the task and the standard deviation.

N1	P0	TX	TX.txt	Need	0.9	0.1
P0	P1	T0	T0.txt	Application	0.9	0.1
P1	P2	T1	T1.txt	Capabilities	0.9	0.1
P2	P3	T2	T2.txt	SysEng	0.9	0.1
P3	P4	T3	T3.txt	SysArchitect	0.9	0.1
P4	P5	T4	T4.txt	SoftEng	0.9	0.1
P5	P6	T5	T5.txt	SoftDev	0.9	0.1

Figure A-15 – Transformations File

Below is the transformation matrix T1. The other transformation matrices are too big and would be unreadable to be shown in pictorial form (see Section 6.2 for details).

0.10	0.08	0.08	0.02	0.10	0.12	0.20	0.10	0.10	0.00	0.00	0.00
0.10	0.08	0.08	0.02	0.10	0.12	0.10	0.10	0.10	0.00	0.20	0.00
0.10	0.08	0.08	0.02	0.10	0.12	0.10	0.10	0.10	0.00	0.00	0.20
0.10	0.08	0.08	0.02	0.10	0.12	0.10	0.00	0.00	0.40	0.00	0.00
0.10	0.08	0.08	0.02	0.10	0.12	0.10	0.00	0.00	0.40	0.00	0.00
0.10	0.08	0.08	0.02	0.10	0.12	0.10	0.00	0.00	0.00	0.00	0.00
0.10	0.08	0.08	0.02	0.10	0.12	0.10	0.00	0.00	0.00	0.00	0.00
0.10	0.08	0.08	0.02	0.10	0.12	0.10	0.00	0.00	0.00	0.00	0.00
0.10	0.08	0.08	0.02	0.10	0.12	0.10	0.00	0.00	0.00	0.00	0.00
0.10	0.08	0.08	0.02	0.10	0.12	0.10	0.00	0.00	0.00	0.00	0.00

Figure A-16 – Transformation File

A.4. Simulation Features

SIAABMSim includes a number of features provided by Repast, among them Multi-Run and Dynamic Class Inspection and Attribute Change.

Multi-Run

Repast provides a mechanism to run multiple simulations in batch known as Multi-Run. Each individual run will take a different random seed. When SIAABMSim is executed in multi-run the output displays are disabled and the results of the simulation are recorded in a text file for subsequent analysis. The parameters recorded include the final duration, effort, effectiveness and efficiency of the acquisition. The MultiRunID in the Simulation Parameters Window is appended to the file name to identify the Simulation Results.

Dynamic Attribute Change

Repast provides a mechanism to inspect classes and modify public attributes when the simulation is in progress, whether ‘running’ or ‘paused’. SIAABMSim uses this feature to allow change to an actor’s attributes during the simulations by clicking on the actor in the Environment window. The screenshot below shows a Manager actor being changed to become ‘aggressive’ during the course of the simulation. Attributes of the organisations and departments can also be changed.

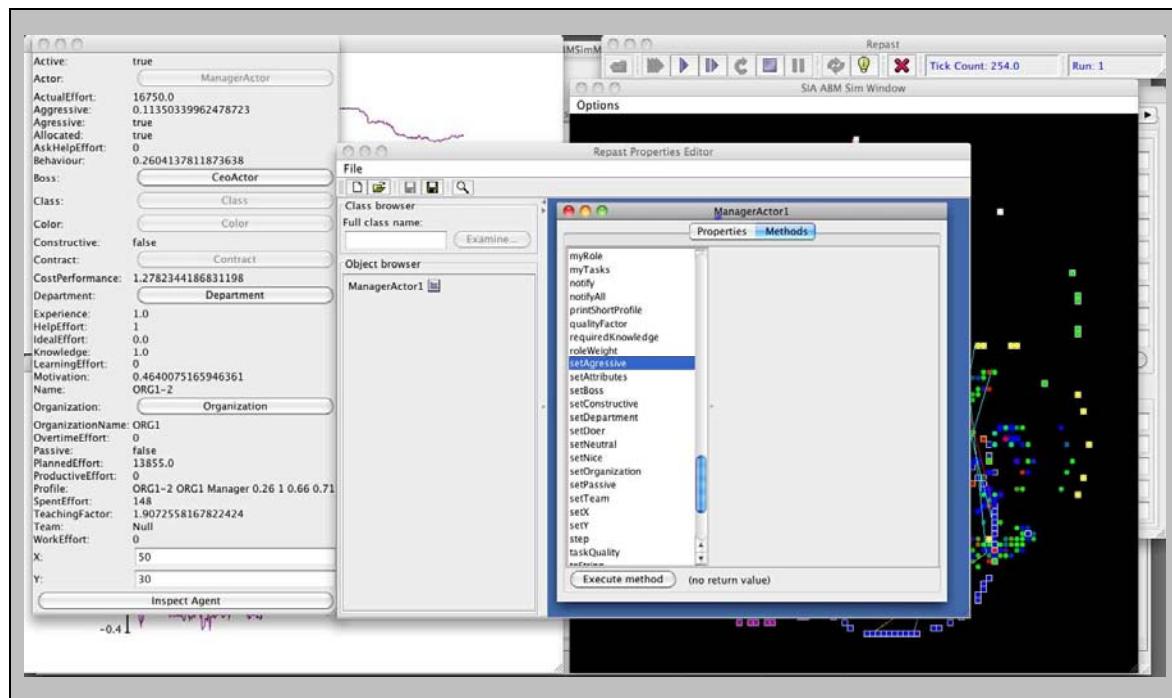


Figure A-17 – Changing Attributes of an Actor during the Simulation

A.5. Configuration Files for Testing Scenarios

SimSettings File

SimSettings-Ideal		SimSettings-Real	
GraphicDisplay	Y	GraphicDisplay	Y
Development	S	Development	S
IterativeFactor	0.0	IterativeFactor	0.5
nTaskIncrements	5	nTaskIncrements	5
TargetEffectiveness	1.0	TargetEffectiveness	1.0
TargetEffort	-1.0	TargetEffort	-1.0
TargetDuration	-1.0	TargetDuration	-1.0
ExplicitReward	0.0	ExplicitReward	0.1
ImplicitReward	0.0	ImplicitReward	0.1
PraiseThreshold	1.0	PraiseThreshold	0.9
ReprimandThreshold	1.0	ReprimandThreshold	1.1
DelayPraise	-1	DelayPraise	100
DelayReprimand	-1	DelayReprimand	100
MinKEM	0.4	MinKEM	0.4
EffortFactor	1.0	EffortFactor	1.0
ReworkEffortFactor	1.0	ReworkEffortFactor	1.0
ReviewFactor	1.0	ReviewFactor	1.0
ReworkFactor	1.0	ReworkFactor	1.0
TaskProgressRate	1.0	TaskProgressRate	1.0
LearningRate	0.0	LearningRate	0.02
ForgettingRate	0.0	ForgettingRate	0.005
ExperienceRate	0.0	ExperienceRate	0.3
BehaviourRate	0.0	BehaviourRate	0.01
ConstructiveRate	0.0	ConstructiveRate	0.05
PassiveRate	0.0	PassiveRate	0.15
AggressiveRate	0.0	AggressiveRate	0.10
MotivationRate	0.0	MotivationRate	0.05
RemoteInteractions	0.0	RemoteInteractions	0.001
InteractionFactor	0.0	InteractionFactor	0.8
LearningFactor	0.0	LearningFactor	0.5
CollaborationFactor	0.0	CollaborationFactor	0.8
IterativeLearningFactor	0.0	IterativeLearningFactor	1.0
FlipBehaviour	0.0	FlipBehaviour	0.001
CopySeniorBehaviour	0.0	CopySeniorBehaviour	0.05

Figure A-18 – SimSettings-Ideal and SimSettings-Real

Probabilities File

Probabilities-Ideal						
Header	Aggressive	Doer	Constructive	Nice	Passive	Neutral
Start	0.0	0.0	0.0	0.0	0.0	0.0
Respond	0.0	0.0	0.0	0.0	0.0	0.0
AskHelp	0.0	0.0	0.0	0.0	0.0	0.0
OfferHelp	0.0	0.0	0.0	0.0	0.0	0.0
AcceptHelp	0.0	0.0	0.0	0.0	0.0	0.0
Help	0.0	0.0	0.0	0.0	0.0	0.0
Praise	0.0	0.0	0.0	0.0	0.0	0.0
Reprimand	0.0	0.0	0.0	0.0	0.0	0.0
Learning	0.0	0.0	0.0	0.0	0.0	0.0
Forgetting	0.0	0.0	0.0	0.0	0.0	0.0
Motivated	0.0	0.0	0.0	0.0	0.0	0.0
Demotivated	0.0	0.0	0.0	0.0	0.0	0.0
Working	1.0	1.0	1.0	1.0	1.0	1.0
Overtime	0.0	0.0	0.0	0.0	0.0	0.0
Aggressive	0.0	0.0	0.0	0.0	0.0	0.0
Constructive	0.0	0.0	0.0	0.0	0.0	0.0
Passive	0.0	0.0	0.0	0.0	0.0	0.0
BAggressive	0.0	0.0	0.0	0.0	0.0	0.0
BConstructive	0.0	0.0	0.0	0.0	0.0	0.0
BPassive	0.0	0.0	0.0	0.0	0.0	0.0
MngAggressive	0.0	0.0	0.0	0.0	0.0	0.0
MngConstructive	0.0	0.0	0.0	0.0	0.0	0.0
MngPassive	0.0	0.0	0.0	0.0	0.0	0.0
TLAGgressive	0.0	0.0	0.0	0.0	0.0	0.0
TLConstructive	0.0	0.0	0.0	0.0	0.0	0.0
TLPassive	0.0	0.0	0.0	0.0	0.0	0.0
QualityMotiv	0.0	0.0	0.0	0.0	0.0	0.0
CostSchedMotiv	0.0	0.0	0.0	0.0	0.0	0.0
ExplicitReward	0.0	0.0	0.0	0.0	0.0	0.0
ImplicitReward	0.0	0.0	0.0	0.0	0.0	0.0
Probabilities-Real						
Header	Aggressive	Doer	Constructive	Nice	Passive	Neutral
Start	0.05	0.10	0.20	0.10	0.05	0.01
Respond	0.50	0.80	0.90	0.80	0.70	0.50
AskHelp	0.05	0.20	0.50	0.30	0.10	0.05
OfferHelp	0.02	0.05	0.80	0.10	0.05	0.01
AcceptHelp	0.30	0.40	0.80	0.50	0.20	0.10
Help	0.05	0.10	0.50	0.30	0.10	0.05
Praise	0.01	0.05	0.30	0.20	0.05	0.0
Reprimand	0.50	0.30	0.05	0.01	0.0	0.0
Learning	0.10	0.20	0.30	0.10	0.05	0.01
Forgetting	0.05	0.10	0.15	0.20	0.25	0.30
Motivated	0.10	0.15	0.20	0.15	0.10	0.15
Demotivated	0.20	0.25	0.40	0.25	0.10	0.15
Working	0.90	0.98	0.95	0.90	0.80	0.85
Overtime	0.70	0.50	0.30	0.10	0.05	0.10
Aggressive	0.50	0.40	0.30	0.20	0.1	0.0
Constructive	0.10	0.20	0.80	0.30	0.10	0.0
Passive	0.03	0.05	0.10	0.15	0.25	0.20
BAGgressive	0.50	0.40	0.30	0.20	0.1	0.0
BConstructive	0.20	0.20	0.5	0.30	0.10	0.0
BPassive	0.0	0.0	0.0	0.0	0.0	0.0
MngAggressive	0.50	0.40	0.30	0.20	0.1	0.0
MngConstructive	0.20	0.20	0.5	0.30	0.10	0.0
MngPassive	0.0	0.0	0.0	0.0	0.0	0.0
TLAGgressive	0.50	0.40	0.30	0.20	0.1	0.0
TLConstructive	0.20	0.20	0.5	0.30	0.10	0.0
TLPassive	0.03	0.05	0.10	0.15	0.25	0.20
QualityMotiv	0.2	0.3	0.7	0.6	0.55	0.5
CostSchedMotiv	0.8	0.7	0.3	0.4	0.35	0.5
ExplicitReward	0.7	0.6	0.3	0.4	0.55	0.5
ImplicitReward	0.3	0.4	0.7	0.6	0.35	0.5

Figure A-19 – Probabilities-Ideal and Probabilities-Real

Organisations Ideal

Organisations-1Org-Ideal-300									
ORG	ORG1	0.1	Application	Capabilities	SysEng	SysArchitect	SoftEng	SoftDev	
CEO	1	0.33	0.0	1.0	0.0	1.0	0.0	1.0	0.0
Manager	20	0.33	0.0	1.0	0.0	1.0	0.0	1.0	0.0
Expert	0	0.33	0.0	1.0	0.0	1.0	0.0	1.0	0.0
TeamLeader	60	0.33	0.0	1.0	0.0	1.0	0.0	1.0	0.0
TeamMember	300	0.33	0.0	1.0	0.0	1.0	0.0	1.0	0.0

Organisations-3Orgs-Ideal-600									
ORG	ORG1	0.2	Nil	Application	SysEng	SysArchitect	SoftEng	SoftDev	
CEO	1	0.33	0.0	1.0	0.0	1.0	0.0	1.0	0.0
Manager	4	0.33	0.0	1.0	0.0	1.0	0.0	1.0	0.0
Expert	0	0.33	0.0	1.0	0.0	1.0	0.0	1.0	0.0
TeamLeader	8	0.33	0.0	1.0	0.0	1.0	0.0	1.0	0.0
TeamMember	36	0.33	0.0	1.0	0.0	1.0	0.0	1.0	0.0
ORG	ORG2	0.2	Capabilities	SysEng	SysArchitect				
CEO	1	0.33	0.0	1.0	0.0	1.0	0.0	1.0	0.0
Manager	18	0.33	0.0	1.0	0.0	1.0	0.0	1.0	0.0
Expert	0	0.33	0.0	1.0	0.0	1.0	0.0	1.0	0.0
TeamLeader	48	0.33	0.0	1.0	0.0	1.0	0.0	1.0	0.0
TeamMember	240	0.33	0.0	1.0	0.0	1.0	0.0	1.0	0.0
ORG	ORG3	0.2	SoftEng	SoftDev					
CEO	1	0.33	0.0	1.0	0.0	1.0	0.0	1.0	0.0
Manager	24	0.33	0.0	1.0	0.0	1.0	0.0	1.0	0.0
Expert	0	0.33	0.0	1.0	0.0	1.0	0.0	1.0	0.0
TeamLeader	66	0.33	0.0	1.0	0.0	1.0	0.0	1.0	0.0
TeamMember	324	0.33	0.0	1.0	0.0	1.0	0.0	1.0	0.0

Figure A-20 – Organisations-Ideal

Appendix B: Refereed Paper 1

Title

‘A Task Model of Software Intensive Acquisitions: An Integrated Tactical Avionics System Case Study’

Authors

Ricardo Peculis, Derek Rogers and Peter Campbell

Source

Refereed paper presented at the Twelfth Australian International Aerospace Congress, AIAC12, held in Melbourne, VIC, Australia, from 19 to 22 March 2007.

Extended Abstract

The acquisition of complex software-intensive systems is fraught with significant risks and often incurs schedule delays, cost overruns and reduced functionality when the product is finally delivered.

This paper presents a model to assess the effectiveness of software-intensive acquisitions, founded on the premise that the solution depends on a series of transformations that transform input products in one domain into output products in another domain. Transformations are performed by people, and require knowledge and skills pertinent to both input and output domains.

The model defines three domains in the acquisition space: Situation Domain, Problem Domain and Solution Domain. A physical situation identifies a need expressed by a set of products in the Situation Domain. The need leads to the definition of the problem as another set of products in the Problem Domain. The solution that resolves the problem comprises a set of products in the Solution Domain.

Ideally, the need should be accurately expressed, the problem well defined and the solution correctly implemented to satisfactorily address the need. In reality, the expressed need diverges from the real need; the expressed problem incurs further

distortions and carries the sum of all these distortions on to the solution. The resulting implemented solution is unlikely to satisfy the real need. Distortions occur because people are limited by their own knowledge, skills, cognition and emotions and are moved by personal and corporate goals and bounded by their roles in the social organisation.

The proposed model represents the products of a domain as vectors in a vector space; and tasks as transformations that change products from one domain into another. The Analytic Hierarchy Process (AHP) is used to determine the input values for the model, to quantify a physical situation so that it can be manipulated by vector mathematics. Transformations require a nominal level of knowledge, skills and effort to be executed without distortions, which occur when the person executing the transformation possesses less than the required knowledge and skills, or has those skills subverted by other factors, reducing the effectiveness of the acquisition. The effect of ‘Chinese Whispers’ is easily demonstrated with the model and is an analogy of the type of distortion that can occur.

A case study based on the acquisition of an integrated tactical and avionics system for the Super Seasprite SH-2G(A) Helicopter is presented to demonstrate how the model proposed in this paper can be applied to a defence software-intensive acquisition. This task model provides the basis for a more comprehensive model that can be applied to understand complex software-intensive acquisitions. Aspects that influence the effectiveness of the acquisition can be explored by the model such as: variations in input parameters and resources, the propagation of lack of knowledge and skills and how the development of software is likely to be affected; how troubled projects are likely to get worse; cause and effect of tension between engineering and project management; individual and corporate motivation; staff morale; learning and cooperation. Finally, the application of the model using agent-based simulations is presented as further work.

A Task Model of Software Intensive Acquisitions: An Integrated Tactical Avionics System Case Study

Ricardo Peculis
Senior Systems Engineer,
Computer Sciences Corporation
PhD Student,
Systems Engineering Evaluation Centre, University of South Australia

Dr Derek Rogers
Adjunct Senior Research Fellow,
Systems Engineering Evaluation Centre, University of South Australia

Prof Peter Campbell
Professor of Systems Modelling and Simulation
Systems Engineering Evaluation Centre, University of South Australia

ABSTRACT

The acquisition of complex software intensive systems is fraught with significant risks and often incurs schedule delays, cost overruns and reduced functionality when the product is finally delivered. This paper presents a model to assess the effectiveness of software intensive acquisitions, founded on the premise that the solution depends on a series of transformations that transform input products in one domain into output products in another domain. Transformations are performed by people, and require knowledge and skills pertinent to both input and output domains. Ideally, products should be transformed without distortions resulting into the desirable solution. In reality, distortions occur because people are limited by their own knowledge, skills, cognition, emotions and are moved by personal and corporate goals and bounded by their roles in the social organisation. The proposed model represents the products of a domain as vectors in a vector space; and tasks as transformations that change products from one domain into another. The Analytic Hierarchy Process (AHP) is used to determine the input values for the model, to quantify a physical situation so that it can be manipulated by vector mathematics. Transformations require a nominal level of knowledge, skills and effort to be executed without distortions, which occur when the person executing the transformation possesses less than the required knowledge and skills, or has those skills subverted by other factors, reducing the effectiveness of the acquisition. The effect of ‘Chinese Whispers’ is easily demonstrated with the model and is an analogy of the type of distortion that can occur. A case study based on the acquisition of an integrated tactical and avionics system for the Super Seasprite SH-2G(A) Helicopter is presented to demonstrate how the model proposed in this paper can be applied to a defence software intensive acquisition. The paper concludes by showing how the proposed task model provides the basis for a more comprehensive model that can be applied to understand, explore and design complex acquisitions.

KEY WORDS

Software Intensive, Acquisition, Social Organisation, Analytic Hierarchy Process, Integrated Tactical Avionics, Case Study, Helicopter.

INTRODUCTION

The acquisition of complex software intensive systems is fraught with significant risks and often incurs schedule delays, cost overruns and reduced functionality when the product is finally delivered. Software intensive systems have a strong dependency on specific and innovative

software products to execute their functions. Without software, a software intensive system becomes inoperative.

This paper presents a model to assess the effectiveness of software intensive acquisitions by comparing the actual results against the ideal. The model is founded on the premise that the solution depends on a series of transformations that transform input products in one domain into output products in another domain. Product transformations are performed by people and require knowledge and skills pertinent to both input and output domains.

The model defines three domains in the acquisition space: Situation Domain, Problem Domain and Solution Domain. A physical situation identifies a need expressed by a set of products in the Situation Domain. The need leads to the definition of the problem as another set of products in the Problem Domain. The solution that resolves the problem comprises a set of products in the Solution Domain.

Ideally, the need should be accurately expressed, the problem well defined and the solution correctly implemented to satisfactorily address the need. In reality, the expressed need diverges from the real need; the expressed problem incurs further distortions and carries the sum of all these distortions on to the solution. The resulting implemented solution is unlikely to satisfy the real need. Distortions occur because people are limited by their own knowledge, skills, cognition and emotions and are moved by personal and corporate goals and bounded by their roles in the social organisation.

The model proposed in this paper represents the products of a domain as vectors in a vector space; and tasks as transformations that change products from one domain into another. The Analytic Hierarchy Process (AHP) is used to determine the input values for the model, to quantify a physical situation so that it can be manipulated by vector mathematics. Transformations require a nominal level of knowledge, skills and effort to be executed without distortions. If the person executing the transformation applies less than the nominal effort, or possesses less than the required knowledge and skills, or has those skills subverted by other factors, distortions occur and reduce the effectiveness of the acquisition. The effect of ‘Chinese Whispers’ is easily demonstrated with the model and is an analogy of the type of distortion that can occur.

A case study based on the acquisition of an integrated tactical and avionics system for the Super Seasprite SH-2G(A) Helicopter being acquired by the Commonwealth of Australia for the Royal Australian Navy (RAN) is presented to demonstrate how the model proposed in this paper can be applied to a defence software intensive acquisition. This task model provides the basis for a more comprehensive model that can be applied to understand complex software intensive acquisitions. Aspects that influence the effectiveness of the acquisition can be explored by the model such as: variations in input parameters and resources, the propagation of lack of knowledge and skills and how the development of software is likely to be affected; how troubled projects are likely to get worse; cause and effect of tension between engineering and project management; individual and corporate motivation; staff morale; learning and cooperation. Finally, the application of the model using agent-based simulations is presented as further work.

THE ACQUISITION MODEL

The aim of the acquisition is to engineer and produce the system that will satisfy the need. The system comprises of “end-products” and “enabling-products” (EIA, 1999). End or operational products are the elements that will be delivered to the ultimate user and that will be applied as the solution that meets the need, being hardware, software, facilities, people and services. The processes required to enable operational products are performed by enabling-products (e.g. development products, test products, training products, production products, support products,

etc). Enabling-products are of value during the development of the operational products and are discarded when the latter enter in operation.

The model is founded on the premise that the result of the acquisition depends on the sum of all operational products, plus the cost and time utilised to produce all operational and enabling-products. The quality of the solution is determined by how well the operational products meet the need when put into operation.

The acquisition model will include products, the tasks to produce these products and the people that will execute the tasks. Products, whether operational or enabling, can be physical or functional in nature. Physical products have physical properties, while functional products are usually ideas, concepts and specification of products that do not exist physically (Aslaksen, 1996). Products can be inputs to tasks to produce other products and both physical and functional products are produced by tasks executed by people.

Modelling the Acquisition through Domain Spaces

Products are described by functional and performance attributes associated with the domain space in which they exist. The acquisition domain space comprises of all operational and enabling-products. Products can be grouped logically in accordance with sub-domains of the acquisition space. Three main sub-domain spaces within the acquisition space are defined: Situation Domain, Problem Domain and Solution Domain.

The acquisition originates in the Situation Domain, where a need exists and can be expressed by a set of products in the Situation Domain. The need leads to the definition of the problem as another set of products in the Problem Domain. The solution that resolves the problem comprises a set of products in the Solution Domain. The transformations of products from need to problem and then to solution require human resources with knowledge and skills specific of each domain.

Both the situation and solution domains contain physical and functional products. The need is perceived and expressed in the form of specifications of functional products. The solution is defined also as functional products, such as specification and design, and implemented by physical products, such as equipment and operational procedures. The problem domain is functional in nature. Ideally, the process should occur as shown on Figure 1, where the need is accurately expressed, the problem is well defined, the solution is correctly implemented and addresses satisfactorily the need.

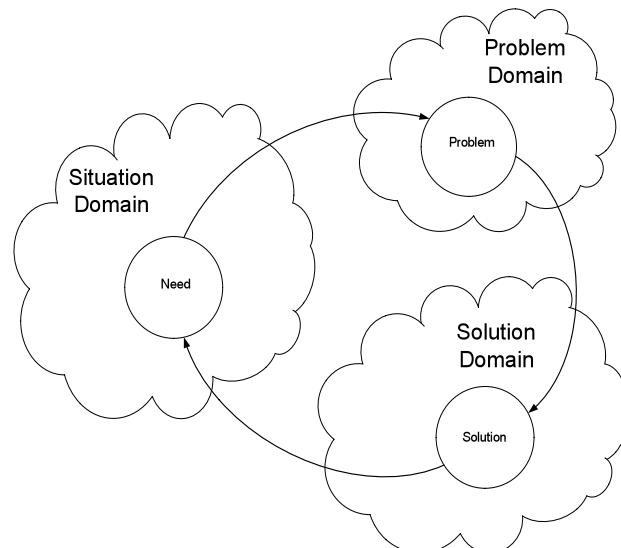


Figure 1 – Transformations between Domain Spaces.

The execution of the tasks that produces the products require knowledge associated with the domain space of the input and output products. The quality of the task execution will depend on people's knowledge and skills and their motivation to execute the task.

Figure 2 shows the real case. Within the Situation Domain, as the "Real Need" is interpreted into the "Perceived Need" they diverge. As the latter becomes the "Expressed Need" it incurs further distortion. Thus, within the Problem Domain, the derived "Expressed Problem" carries the sum of all these distortions on to the Solution Domain. The resulting "Solution Implemented" is quite unlikely to satisfy the "Real Need".

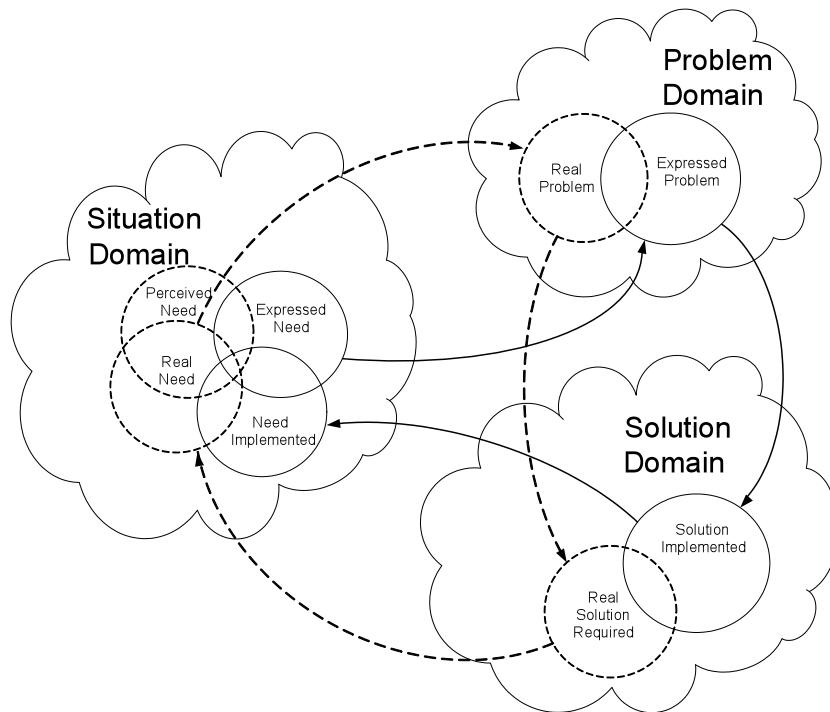


Figure 2 – Actual transformations between Domain Spaces.

Distortions occur because people within social organisations, limited by their own knowledge, skills, cognition, emotions, moved by personal and corporate goals and bounded by their roles in the social organisation, are responsible for the execution of transformations between and within domains.

The ideal transformations should accurately move from one domain to another without distortions, leading to a solution that satisfactorily improves the situation. Success thus depends on how well the transformations occur.

Effectiveness and Efficiency of Software Intensive Acquisitions

Effectiveness indicates how well the acquisition achieves its goals and efficiency represents how much waste is produced in the process. Hitchins (1992) suggests that effectiveness should be assessed comparing the real system against the ideal. From Figure 2, effectiveness indicates how close the implemented solution came to fulfil the real need, i.e. the unbroken line circles coinciding with the dotted circles, and efficiency shows how much effort, in the form of enabling tasks and rework, is required to bring the implemented solution to meet the real need.

In the ideal system people possess knowledge and skills required to execute the transformation tasks, while in the real system this condition will be less than ideal. Lack of knowledge and skills, or lack of applying knowledge and skills available, are some of the causes of low effectiveness and efficiency in software-intensive acquisitions.

THE TASK MODEL

The ACTS theory (Carley and Prietula, 1994) sets the context for the task model. In accordance with the ACTS theory, organisations are collections of intelligent agents cognitively restricted, task oriented and socially situated. In the acquisition of complex systems, people are oriented to apply their knowledge, skills and perceptions of the situation in a social environment to perform the task of expressing the need, formulating the problem, establishing constraints and finding and implement a solution that satisfactorily meets the need.

The task model comprises of products and tasks. Products are functional or physical objects that together constitute the whole. Tasks are the transformations applied to products to produce other products within the same or in other domains. The execution of transformation tasks requires a nominal level of knowledge and skills associated with input and output products and their respective domains.

A nominal effort is associated to the execution of a task. If an ideal agent applies less than the nominal effort, the distortions on the output product will be proportional to the ratio between the actual and nominal task effort.

Although the example adopts linear transformations, the system is not expected to be linear. Software-intensive acquisitions are complex adaptive systems and as people learn and adapt, knowledge and skills are not constant throughout the course of the acquisition. Learning and cooperation can improve useful knowledge. Learning and the application of available knowledge and skills are fuelled by motivation. Divergent motivation can produce non-linear unwanted effects, which in turn creates tension between engineering management and project management (Aslaksen, 1996), and are likely to worsen the effectiveness of the system.

Products and Tasks in the Acquisition Space

The need identifies capabilities that are realised by products organised hierarchically. A process of analysis is a transformation task that identifies the products to fulfil the desired capabilities.

The need identifies the problem requiring a solution and another transformation, the process of engineering, identifies the solution to the problem. The need is communicated through the concept of execution and functional performance documents, which drive the specification, design and construction of the products that will form the solution. The solution is also realised by transformation tasks in the form of the application of engineering knowledge, processes and skills. The execution of transformation tasks requires knowledge of input and output products. Lack of knowledge will lead to distortions and omissions in output products.

The solution is likely to include products realised by hardware, software and operational procedures. As the interest of the model is in the implications of the development of software, the model will focus on software products. The development of software, like any physical product, requires specification, design and construction.

The specification and design of software components depend on systems specification and design documents. If the software specification and design are complete, the software development activity will not require any additional information from previous domains. The need for missing information would otherwise flow into the software domain. Lack of information, knowledge and expertise on capabilities and products identified in previous

domains while the software is in production is one of the most common reasons of software delays and cost increases, although not always recognised.

Amongst the root causes for software project failure are inaccurate estimation and changing requirements (Jones 1995, 2006; Standish 1998). Lack of knowledge and skills are drivers of poor specification and estimation and can be associated with the root causes of software project failure. There is a lack of engineering knowledge and skills to specify software components and lack of management knowledge and skills to recognise, estimate, plan and manage this deficiency.

Virtual Products and Tasks in the Acquisition Space

In a real acquisition, products and tasks are defined by their functional and physical attributes in the form of requirements, specifications, conditions of operation and processes, expressed in ways to be manipulated by people, that is, textually, graphically and verbally.

The proposed model is intended for simulations in a computational environment, where virtual agents, representing people, teams and organizations, manipulate virtual products and tasks. It would be difficult, if possible at all, to implement agents that would be able to discuss, analyse, write, interpret, implement and verify textual requirements. For that reason, products and tasks will be represented numerically.

Products will be defined on a functional hierarchy and expressed in numerical form. The Analytic Hierarchy Process (AHP) (Saaty, 2000) is used to determine the numeric values based on the contribution of each product to the whole.

Tasks transform products from one level to other products in the next level down. The execution of the transformation tasks requires a nominal level of human resources in the form of knowledge, skills and effort. The nominal knowledge and skills depend on the parent product and its derived sub-products, and effort determines the duration of the task executed by a single agent that possesses at least the nominal knowledge and skills. When an ideal agent applies the nominal effort, the task is executed without distortions and the product is correctly produced, otherwise distortions occur.

Analytic Hierarchy Process

The Analytic Hierarchy Process (AHP) is a framework of multi-valued logic based on the innate human ability to use information and experience to construct ratio scales through paired comparison (Saaty, 2000). The object of the analysis is arranged on a hierachic network structure that brakes down the whole into its parts thus allowing paired comparison. Paired comparison is done using “The Fundamental Scale” of nine levels 1-9, indicated in Table 2.

The objective of the AHP is to pair compare all components in the system of interest to determine the weight of importance or contribution of each component to the whole.

Table 2 – The Fundamental Scale of AHP

Intensity of Importance	Definition	Explanation
1	Equal importance	The two components contribute equally to the objective
2	Weak importance	
3	Moderate importance	Experience and judgement slightly favour one component over the other
4	Moderate plus importance	
5	Strong importance	Experience and judgement strongly favour one component over the other
6	Strong plus importance	
7	Very strong importance	One component is favoured very strongly over another; its dominance demonstrated in practice
8	Very, very strong importance	
9	Extreme importance	The evidence favouring one component over another is of the highest possible order of affirmation

If the system of interest has “n” components, the pair comparison is obtained by an $n \times n$ square matrix called the Priority Matrix. The weigh of importance of each of the “n” components is given by the normalised principal eigenvector, obtained from the maximum eigenvalue of the Priority Matrix.

The Acquisition as Vectors in a Vector Space

Products are modelled as vectors in a Euclidean Vector Space, while tasks are transformations that create other products as vectors within the same or into other vector spaces. The numeric value of a product represents its contribution to the whole. The complexity of a product is not determined by the absolute value of its numeric representation, but it is associated with its relationship with other products and by the knowledge, skills and effort required to produce the product.

The number of linearly independent products required to represent the acquisition determines the dimension of the acquisition space. The acquisition space adopts the standard base, comprising of unitary orthogonal vectors corresponding to the dimensions of the acquisition space.

Products are transformed into other products through tasks executed by the agents. Tasks are modelled as linear transformations matrices, plus the knowledge and effort required to performing the task.

Useful Knowledge

The effectiveness of the execution of the tasks depends of the agent’s useful knowledge and is proportional to the projection of the agent’s knowledge/skills vector onto the task’s knowledge/skills vector, shown on Figure 3. The normalised value of 1.0 represents the required knowledge/skills.

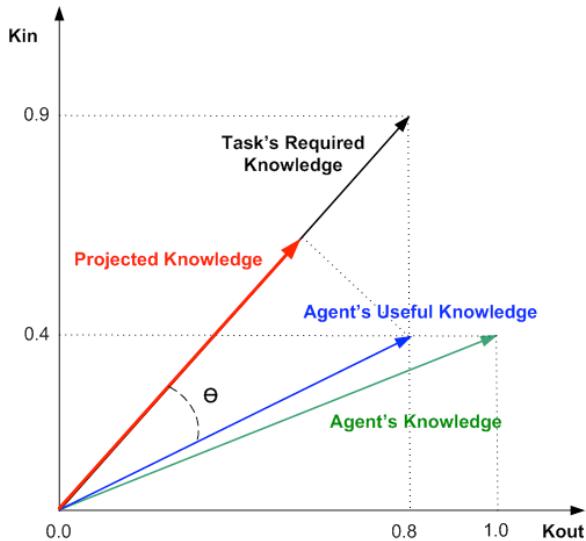


Figure 3 – Useful Knowledge.

Effective Knowledge = | Agent's Useful Knowledge| cos θ , where

$$\theta = \arccos\left(\frac{a \cdot b}{|a||b|}\right) \text{ the angle between the two Task's Knowledge and Agent's Useful Knowledge vectors.}$$

The agent's useful knowledge is limited by the amount of knowledge required by the task on each dimension; otherwise the excess of knowledge/skills in one dimension would compensate the lack of knowledge/skills in another dimension. The effectiveness of the task execution is proportional to the projections of the agent's useful knowledge into the task's knowledge/skills required.

APPLICATION CASE STUDY

This illustrative example is a case study based on the specification of the Super Seasprite SH-2G(A) Helicopter being acquired by the Commonwealth of Australia for the Royal Australian Navy (RAN). The first two levels of decomposition, comprising the roles and capabilities of the aircraft, are based on public information (DMO, 2006; NOC, 2006; Scott et al., 2004). The relative level of importance of roles and capabilities are hypothetical. The model that follows represents the ideal scenario shown on Figure 1.

The Need

The Royal Australian Navy identified the need for a helicopter to support operations from the ANZAC class frigates. Two primary roles for the aircraft were identified:

- a. Increase the ship's effectiveness by expanding surveillance capability;
- b. Contribute to the ship's combat capabilities providing anti-surface and anti-submarine warfare.

The secondary roles for the aircraft comprise of supporting missions such as search-and-rescue and medical evacuation and crew training. The need also determines that the aircraft is to be operated by a crew of two, day and night and under adverse weather. The AHP was used to estimate the hypothetical relative contribution of the aircraft roles, shown on column P_1 in Table 3.

Table 3 – Aircraft Roles

Aircraft Roles	P_1	SV	ASH	ASB	SR	SM	TS
Surveillance (SV)	0.41	1	3	3	7	9	9
Anti-Ship Warfare (ASH)	0.28	1/3	1	3	7	9	9
Anti-Submarine Warfare (ASB)	0.18	1/3	1/3	1	5	7	9
Search and Rescue (SR)	0.07	1/7	1/7	1/5	1	3	7
Support Missions (SM)	0.04	1/9	1/9	1/7	1/3	1	5
Training Support (TS)	0.02	1/9	1/9	1/9	1/7	1/5	1

The need defines six roles (Table 3) and twelve capabilities: HMI for crew of two (HMI); Aircraft Monitor & Supervision (MS); Mission Preparation Support (MPS); Mission Debrief Support (MDS); Electronic Navigation (NAV); Radio Communications (COM); Surveillance Sensors (SVS); Electronic Warfare (EW); Tactical Data Management (TDM); Tactical Navigation (TNM); Anti-Ship Weapons (WAS); and Anti-Sub Weapons (WASb). Table 4 shows the need in the form of the dependency between roles and capabilities.

Table 4 – The Need

Roles		Capabilities												
		P_2	HMI	AMS	MPS	MDS	NAV	COM	SVS	EW	TDM	TNM	WAS	WASb
SV	0.41		0.10	0.08	0.08	0.02	0.10	0.12	0.20	0.10	0.10	0.10	0.00	0.00
ASH	0.28		0.10	0.08	0.08	0.02	0.10	0.12	0.10	0.10	0.10	0.00	0.20	0.00
ASB	0.18		0.10	0.08	0.08	0.02	0.10	0.12	0.10	0.10	0.10	0.00	0.00	0.20
SR	0.07		0.10	0.08	0.08	0.02	0.10	0.12	0.10	0.00	0.00	0.40	0.00	0.00
SM	0.04		0.10	0.08	0.08	0.02	0.10	0.12	0.10	0.00	0.00	0.40	0.00	0.00
TS	0.02		0.10	0.08	0.08	0.02	0.10	0.12	0.10	0.08	0.08	0.08	0.08	0.08

The shaded column P_1 represents the hypothetical contribution of the aircraft's roles, as shown on Table 3. The set of non-shaded numbers defines a 6x12 transformation matrix T_1 and represents the contribution of each capability to the aircraft's roles. Although the numbers in T_1 were estimated without AHP, AHP could have been used to better accuracy (each row requires a 12x12 Priority Matrix). The shaded row P_2 represents the hypothetical contribution of each capability to the need as a whole, calculated by equation (1) below, where P_1 and P_2 are respectively the input and output product vectors and T_1 is the transformation matrix:

$$(1) P_2 = P_1 \cdot T_1$$

The elements of the transformation matrix are tasks whose execution is affected by the ratio of the agent's useful knowledge and the knowledge required to execute it. It is interesting to observe that there are multiple dependencies between roles and capabilities. If Surveillance Sensors (SVS) is partially or not available, for example, all roles defined for the aircraft will be affected. Also, to express the need requires knowledge in multiple areas. To specify the need for Surveillance Sensors (SVS) requires not only knowledge of surveillance sensors but also on all of the aircrafts' roles. Lack of knowledge would distort the tasks, i.e. the numbers of the transformation matrix, and cause deficiencies in expressing the need.

The expressed need is presented in the form of an Operational Concept or similar document, represented by equation (1).

The Problem

The expressed need is the input to the next level of decomposition. The problem is expressed in the form of what is required to resolve the expressed need. Capabilities endure further analysis to reveal products required to resolve the problem. The analysis continues until all products and their relationship are completely defined and specified. There are 26 primary products identified for the SH2G(A) helicopter, comprising of HMI devices, Main Data Processor, Mission Data Loader/Recorder, Navigation Devices, Communication Equipment, Aircraft Sensors, Radar, Forward Looking Infra Red (FLIR), Electronic Warfare (EW), Threat Warning, Counter Measure Dispenser (CMDS), Link-11 and various Weapons.

Capabilities are expanded into primary products and their contribution to the whole is estimated. Being P_2 and P_3 respectively the Capabilities vector input and the Primary Products vector output, the transformation matrix T_2 is a 12x26 matrix such that:

$$(2) P_3 = P_2 \cdot T_2$$

The problem analysis is completed when the solution is defined and proposed in the form of a Function Performance Specification, or similar document, for an Integrated Tactical Avionics System, represented by equation (2).

The Solution

The analysis of the problem and the proposed solution are the input to the next transformations. The solution comprises a series of transformation that produce enabling and operational products. Enabling products are the specification and design for the system, hardware and software, while operational products are hardware, software and procedures that will be delivered with the final solution.

Each of the 26 primary products identified in the proposed solution requires a system specification and design, as well as specification and design for human and hardware interfaces. The output of specification and design is then 156 enabling products. The specification and design process is represented by equation (3), where P_3 is the Product input vector, P_4 is the Specification and Design output vector and T_3 is the 26x156 transformation matrix.

$$(3) P_4 = P_3 \cdot T_3$$

The system under development calls for the integration of products that require the development of dedicated software. As the focus of this paper is on software intensive systems, the transformations that follow address only software products.

Each of the 26 primary products depends on software and requires software specification and design. The output of software specification and design is then 52 enabling products. The specification and design process is represented by equation (4), where P_4 is the System Specification and Design input vector, P_5 is the Software Specification and Design output vector and T_4 is the 156x52 transformation matrix.

$$(4) P_5 = P_4 \cdot T_4$$

The specification and design of software products should contain all information required to enable the production of software components. Each of the 26 primary products requires the

development of dedicated software products. The output of the software productions is then 26 operational products. The production of software products is represented by equation (5), where P_5 is the Software Specification and Design input vector, P_6 is the Software products output vector and T_5 is the 52x26 transformation matrix.

$$(5) P_6 = P_5 \cdot T_5$$

System specification and design documents intend to bring knowledge and information from previous domains into the software domain. When this intent is not achieved, lack of information and knowledge will propagate into the software domain and will carry distortions and omissions from specifications created on previous domains. To detect and correct these anomalies requires knowledge beyond the software domain, which usually is no longer or easily available.

Effectiveness and Efficiency

The effectiveness of the solution is given by the sum of the contribution of the end products, which in the ideal case is 1.0. Equation (6) represents the effectiveness of the solution, where in this example p_i are the 26 components of the vector P_6 :

$$(6) Effectiveness = \sum_{i=1}^{n=26} p_i$$

Unless distortions are reduced, whether at the end of the development cycle in the form testing and rework, or as an integral part of the engineering process, 100% effectiveness cannot be attained.

Equation (7) represents the efficiency of the system as the ratio between the actual and ideal effort, where actual effort includes the effort spent to produce enabling and end products plus the effort spent on additional enabling tasks and rework:

$$(7) Efficiency = \frac{\sum_{n=1}^{n=6} ActualEffort(Pn)}{\sum_{n=1}^{n=6} IdealEffort(Pn)}$$

The ‘Chinese Whispers’ Effect

The example shows that to produce 26 software operational products requires development of 252 enabling products in the form of operational concept, functional performance and other specification and design documents. Test products do not impact the solution directly and were not included in the example.

Each element of the transformation matrices, if not zero, is a task that requires knowledge of input and output products, spread over 300 areas of knowledge. Equations (1) to (5) represent five transformations required to produce software products. In summary, to produce 26 outputs needed some 250 intermediate outputs in a five-stage process requiring over 900 tasks. The distortion level of each transformation will determine how correct the end product will be if compared with the ideal solution. This reflects the ‘Chinese Whisperers’ effect caused by the propagation of distortions and omissions due to lack of useful knowledge. Distortions will also occur if less than the nominal effort is applied. Wrong estimation can be associated to lack of knowledge and skills and contribute to reduce the effectiveness of the system.

Distortions and rework caused by lack of useful knowledge, whether in engineering or management, could partially explain schedule delays and cost overrun in software intensive projects.

CONCLUSION

The proposed task model represents the ideal products and transformations found in software intensive acquisitions. Products are represented numerically by their respective contribution to the whole estimated using the AHP. Tasks are matrices that transform input products into output products, and require a nominal level of knowledge, skills and effort. If less than the required knowledge, skills and effort are applied, output products will contain distortions and omissions. Output products are input products of subsequent transformation and distortions propagate in the form of ‘Chinese Whispers’ effect.

Lack of knowledge and skills, can be associated with failure of software intensive acquisitions. Knowledge is acquired and shared through learning and cooperation, and these are driven by motivation. Software intensive acquisitions are complex adaptive systems, and the factors that influence knowledge, are affected by the system and vary during the course of the acquisition.

A case study based on the specification of the Super Seasprite SH-2G(A) Helicopter was presented to demonstrate how the proposed task model can be applied to software intensive acquisitions.

The proposed task model is a component of a more comprehensive socio-organisational model founded on the ACTS theory, and will be applied to agent-based simulations to explore non-linearities and factors of success and failure of complex software intensive acquisitions. Simulation can be used to investigate how the effectiveness of the system is affected by variations in input parameters and by the engineering process models (e.g. waterfall versus incremental). Simulations can also help to understand how software components are likely to be affected by the propagation of lack of useful knowledge and inadequate estimation of effort; how troubled projects are likely to get worse; cause and effect of tension between engineering and project management; and how individual and corporate motivation, staff morale, learning and cooperation affect the effectiveness of the system in meeting the need.

REFERENCES

- Aslaksen, E. W. 1996, *The Changing Nature of Engineering*, McGraw-Hill Australia, Sydney.
- Carley, K. M. and M. J. Prietula 1994, *ACTS Theory: Extending the model of bounded rationality*, In Carley, K. M. and M. J. Prietula (Eds.), Computational Organization Theory. Hillsdale, NJ: Lawrence Erlbaum Associates.
- DMO, 2006, *Project SEA 1411, ANZAC Ship Helicopter*, Defence Materiel Organisation, <http://www.defence.gov.au/dmo/asd/sea1411/sea1411.cfm> viewed on 12 June 2006.
- EIA, 1999, *EIA-632 Standard: Processes for Engineering a System*, Electronic Industries Alliance.
- Hitchins, D. 1992, *Putting Systems to Work*, John Willey & Sons, London, England.
- Jones, C. 1995, *Patterns of Software System Failure and Success*, International Thomson Computer Press, Boston, MA, December 1995.

Jones, C. 2006, *Social and Technical Reasons for Software Project Failures*, Cross Talk, The Journal of Defense Software Engineering, June 2006, p.4-8.

NOC, 2006, The SH-2G(A) Kaman Super Seasprite, Naval Officers Club, <http://www.navalofficer.com.au/seasprite.htm>, viewed on 12 June 2006.

Saaty T. L. 2000, *Fundamental of Decision Making and Priority Theory with The Analytic Hierarchy Process*, Vol. VI of the AHP Series, RWS Publications, Pittsburgh, PA.

Scott, R, Holdanowicz , G, Bostock, I, 2004, *Second coming for the Super Seasprite*, The Aviation Forum, <http://forum.keypublishing.co.uk/showthread.php?t=24443>, viewed on 13 June 2006.

Standish Group 1998, *CHAOS: A receipt for success*, Standish Group, http://www.standishgroup.com/sample_research/PDFpages/chaos1998.pdf, viewed 13 Nov. 2003

Appendix C: Refereed Paper 2

Title

'Using Simulation to Support the Design of Software Intensive Acquisitions'

Authors

Ricardo Peculis, Derek Rogers and Peter Campbell

Source

Refereed paper presented at the Simulation Technical Conference, SimTecT 2007, held in Brisbane, QLD, Australia, from 4 to 7 June 2007.

Extended Abstract

The acquisition of complex software-intensive systems is fraught with significant risks and often incurs schedule delays, cost overruns and reduced functionality when the product is finally delivered. The difficulty in succeeding in the acquisition of software-intensive systems can be associated with the behaviour of a social-technical complex adaptive system (CAS) whose success depends on how people and organisations interact and adapt to achieve their goals, not always in the best interest of the system as a whole.

The aim of the acquisition is to produce a solution that satisfies a need. Software-intensive systems rely on the proper development of software products that depend on other enabling products, such as specifications and design documents, which bring information and knowledge from application and engineering domains into software. Distortions and omissions on enabling products propagate into software as defects, whose detection and correction originate rework, increase costs, cause delays and ultimately failure of software-intensive projects.

Distortions and omissions are caused by lack of knowledge and skills. Motivation, whether individual or corporate, impacts on decision-making and on the application of existing knowledge, learning and cooperation. The influence of behavioural styles, motivation and knowledge creates a non-linear dynamic effect on project performance

that is difficult to predict and control, and is an important aspect to be taken into consideration to design software-intensive acquisitions with better chances of success.

Computer modelling and simulation is indicated to study the behaviour of complex adaptive systems and can be extended to the learning of human systems and the impact of management in the life of organisations. Simulation can be used as a learning environment that allows managers to experiment and try alternative solutions without the drawback of trial and error in real systems.

This paper presents a model based on the ACTS (Agent, Cognition, Task, Social) theory applied to agent-based simulations to investigate the influence of motivation, knowledge and behavioural styles to the success of software-intensive acquisitions. The model is founded on the premise that the solution depends on a series of transformations that transform input products in one domain into output products in another domain. Transformations are performed by people, and require knowledge and skills pertinent to both input and output domains. Ideally, products should be transformed without distortions resulting in the desirable solution. In reality, distortions occur because people are limited by their own knowledge, skills, cognition and emotions; are motivated by personal and corporate goals and bounded by their roles in the social organisation.

An agent-based simulation of a hypothetical software-intensive acquisition is presented to demonstrate non-linear effects of the dynamic interaction between motivation and knowledge, and how simulations can aid to comprehend the complexity of software-intensive acquisitions.

The paper concludes by showing how the proposed model provides the basis for more comprehensive simulations as learning tools that can support the exploration of better designs of software-intensive acquisitions.

Using Simulation to Support the Design of Software Intensive Acquisitions

Ricardo Peculis¹; Derek Rogers²; Peter Campbell³

¹Computer Sciences Corporation; SEEC University of South Australia,
ricardo.peculis@postgrads.unisa.edu.au

²SEEC University of South Australia, *derek.rogers@unisa.edu.au*

³SEEC University of South Australia, *peter.campbell@unisa.edu.au*

Abstract. The acquisition of complex software intensive systems for defence is fraught with significant risks and often incurs schedule delays, cost overruns and reduced functionality when the product is finally delivered. The difficulty in succeeding in the acquisition of software intensive systems can be associated with the behaviour of a socio-technical complex adaptive system (CAS) whose success depends on how people and organisations interact and adapt to achieve their goals. Software intensive systems rely on the proper development of software products that depend on other enabling products, which bring information and knowledge from application and engineering domains into software. Distortions and omissions on enabling products are caused by lack of knowledge and skills and propagate into software as defects, whose detection and correction increase costs, cause delays and in the worst case the failure of software intensive projects. The influence of interaction styles, motivation and knowledge creates a non-linear dynamic effect on project performance that is difficult to predict and control. Computer modelling and simulation is indicated to study the behaviour of complex adaptive systems. Simulation also can be used as a learning environment that allows managers to experiment and try alternative solutions without the drawback of trial and error in real systems. This paper presents a model based on the ACTS (Agent, Cognition, Task, Social) theory applied to agent-based simulations to investigate the influence of motivation, knowledge and interaction styles to the success of software intensive acquisitions. An agent-based simulation of a hypothetical naval helicopter acquisition is presented to demonstrate how non-linear effects arise from the dynamic interaction of people. Results from simulation confirm that lack of knowledge and non-constructive behaviours increase the duration of software projects and reduce the effectiveness of the final product.

1. INTRODUCTION

The acquisition of complex software intensive systems for defence is fraught with significant risks and often incurs schedule delays, cost overruns and reduced functionality when the product is finally delivered. The difficulty in succeeding in the acquisition of software intensive systems can be associated with the behaviour of a socio-technical complex adaptive system (CAS) whose success depends on how people and organisations interact and adapt to achieve their goals, not always in the best interest of the system as a whole.

The aim of the acquisition is to produce a solution that satisfies a need. Software intensive systems rely on the proper development of software products that depend on other enabling products, such as specifications and design documents, which bring information and knowledge from application and engineering domains into software. Distortions and omissions on enabling products propagate into software as defects, whose detection and correction increase costs, cause delays and in the worst case the failure of software intensive projects.

Distortions and omissions are caused by lack of knowledge and skills. Motivation, whether individual or corporate, impacts on decision-making and on the application of existing knowledge, learning and cooperation. The influence of interaction styles, motivation and knowledge creates a non-linear dynamic effect on project performance that is difficult to predict and control, and is an important aspect to be taken into consideration to design the socio-organisation of software intensive acquisitions with better chances of success.

Computer modelling and simulation is indicated to study the behaviour of complex adaptive systems [5] and can be extended to the learning of human systems and the impact of management in the life of organisations [10]. Simulation can be used as a learning environment that allows managers to experiment and try alternative solutions without the drawback of trial and error in real systems. Stacey [10] suggests that simulation is more adequate to understand the dynamics of human systems than traditional methods that adopt interviews and surveys, which do not reveal what is really happening because people in organisations say one thing when they are doing another and usually do not even know what they are doing or why. Simulation can be free of misleading information and allows researchers to hypothesise and test their hypothesis without interference or interfering on the system. However simulation makes simplified assumptions and conclusions are only as good as the fidelity of the model.

This paper presents a model based on the ACTS (Agent, Cognition, Task, Social) theory [1] applied to agent-based simulations to investigate the influence of motivation, knowledge and interaction styles to the success of software intensive acquisitions. The model is founded on the premise that the solution depends on a series of transformations that transform input products from one domain into output products in another domain. Transformations are performed by people, and require knowledge and skills pertinent to both input and output domains. Ideally, products should be transformed without distortions resulting in the desirable solution. In reality, distortions occur because people are limited by their own knowledge, skills, cognition and emotions; are motivated by personal and corporate goals and bounded by their roles in the organisation.

An agent-based simulation of a hypothetical naval helicopter acquisition is presented to demonstrate how non-linear effects arise from the dynamic interaction of people driven by their own motivation, interaction styles and sometimes with less than the ideal knowledge to execute the task. Results from simulation confirm that lack of knowledge and non-constructive behaviours increase the duration of software projects and reduce the effectiveness of the final product.

The paper concludes showing that simulation can aid in comprehending the complexity of software intensive acquisitions, and that the proposed model can be used as a learning tool to support the exploration of better socio-organisational designs of software intensive acquisitions.

2. THE ACQUISITION MODEL

The aim of the acquisition is to engineer and produce the products that will satisfy the need. The acquisition originates in the Situation Domain, where a need exists and can be expressed by a set of products in the Situation Domain. The need leads to the definition of the problem as another set of products in the Problem Domain. The solution that resolves the problem comprises a set of products in the Solution Domain. The acquisition is then represented by products and transformations within the acquisition space [6]. The transformations of products from need to problem and then to solution require human resources with knowledge and skills specific of each domain. The execution of the tasks that produces the products require knowledge associated with the domain space of the input and output products. The quality of the task execution will depend on people's knowledge and skills and their motivation to execute the task.

The need is perceived and expressed in the form of specifications of functional products. The solution is defined also as functional products, such as specification and design, and implemented by physical products, such as equipment and operational procedures. The problem domain is functional in nature. Ideally, the process should occur without distortions. The need should be accurately expressed, the problem well defined, the solution correctly implemented to address satisfactorily the need. In reality, shown on Figure 1, within the Situation Domain, as the "Real Need" is interpreted into the "Perceived Need" they diverge. As the latter becomes the "Expressed Need" it incurs further distortion. Thus, within the Problem Domain, the derived "Expressed Problem" carries the sum of all these distortions on to the Solution Domain. The resulting "Solution Implemented" is quite unlikely to satisfy the "Real Need".

Distortions occur because people within social organisations, limited by their own knowledge, skills, cognition, emotions, moved by personal and corporate goals and bounded by their roles in the social organisation, are responsible for the execution of transformations between and within domains.

The ideal transformations should accurately move from one domain to another without distortions, leading to a solution that satisfactorily improves the situation. Success thus depends on how well the transformations occur.

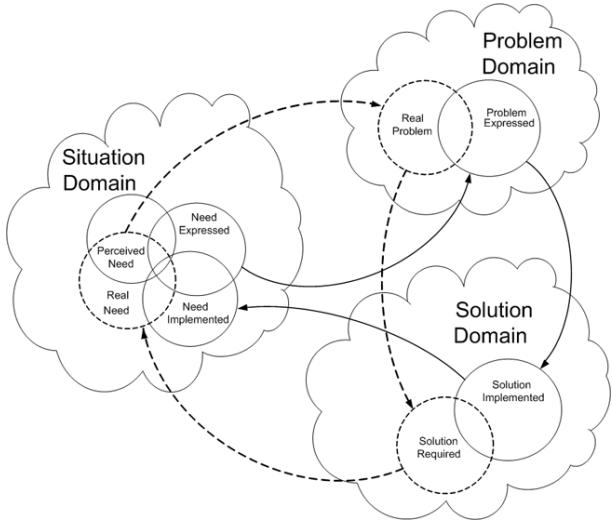


Figure 1: Transformations between Domain Spaces.

2.1 Effectiveness and Efficiency

Effectiveness indicates how well the acquisition achieves its goals and efficiency is measured by how much waste is produced in the process. Hitchins [3] suggests that effectiveness should be assessed comparing the real system against the ideal. From Figure 1, effectiveness indicates how close the implemented solution came to fulfilling the real need, i.e. the unbroken line circles coinciding with the dotted circles, and efficiency shows how much effort, in the form of enabling tasks and rework, is required to bring the implemented solution to meet the real need. Lack of knowledge and skills available, or lack of applying knowledge and skills available, are some of the causes of low effectiveness and efficiency in software-intensive acquisitions.

2.2 Problem-Solving Groups

Problem-solving groups have two principal objectives: (1) quality of results, that requires the maximum utilisation of resources brought by each individual member of the group; and (2) acceptance of the solution, that implies a high level of motivation for carrying out the group's decision [4]. These objectives create pressure on each member of the group to decide between bringing their unique, and possibly controversial, contribution to the problem-solving task, or to conform with group's decisions [4].

The effectiveness of problem-solving groups depends on the group's interaction style, which can be constructive, aggressive or passive [2]. Groups with a constructive style tend to maximise the contribution of each group member, fostering cooperation between the group members to achieve both individual and group objectives. Members of groups with a passive style behave in a ways that promote their security and acceptance, avoiding conflict, limiting information sharing, and allowing themselves to become dominated by the group. In aggressive group style, members of the group will behave to promote their status and position to fulfil their own needs; members also tend to demonstrate their competence doing things perfectly, without seeking help and helping others and losing sight of the group's goals.

Cooke [2] demonstrates that across problem-solving groups, solution quality increases when the group shows

a constructive interactive style and decreases with passive interaction style; and the acceptance of the solution increases with constructive interaction style and decreases with both passive and aggressive interaction styles. Aggressive interaction was shown to be unrelated to the quality of the solution.

3. THE ACTS MODEL

The ACTS theory [1] sets the context for the acquisition model and computer simulation. In accordance with the ACTS theory, organisations are collections of intelligent agents cognitively restricted, task oriented and socially situated. In the acquisition of complex systems, people are oriented to apply their knowledge, skills and perceptions of the situation in a social environment to a problem-solving group activity, comprising of expressing the need, formulating the problem, establishing constraints and finding and implement a solution that satisfactorily meets the need.

3.1 The Task Model

The Task model represents products and tasks within the acquisition space. Products are modelled as vectors in a Euclidian Vector space, while tasks are transformations that created other products as vectors within the same or into other vector spaces [6]. The numeric value of products represents their contribution to the whole, determined by the Analytic Hierarchy Process [9] or equivalent method.

Products are created by transforming an input product P_i by a transformation matrix T_i , as indicated by equation (1):

$$(1) P_{i+1} = P_i \cdot T_i$$

An acquisition with n products is represented by equation (2):

$$(2) P_n = P_1 \cdot \prod_{i=1}^{i=n-1} T_i$$

P_1 is the “need” and P_n is the “solution” product that will satisfy the “need”.

Each element of a transformation matrix is a task that requires a nominal knowledge and effort to be executed. The knowledge required contains components of input and output products. If the person, or virtual agent, assigned to the task possesses at least the nominal knowledge on input and output products and applies the nominal effort required, the task is executed without distortions. Otherwise, the task is executed with distortions that are proportional to the lack of knowledge and may take longer if the person decides to apply effort to learn and acquire the required knowledge.

3.2 The Agent Model

The Agent model represents the people that assign, execute and monitor progress of tasks. Agents have their own knowledge, motivation, interaction style, role and responsibilities. Figure 2 shows the structure of the Agent model in the context of the Social model.

An ideal agent possesses perfect knowledge in all the required areas and would execute tasks without

distortions. A more realistic agent possesses a limited amount of knowledge, determined by its speciality and its experience.

The agent interaction style is classified as constructive, passive or aggressive. Constructive behaviour fosters learning and cooperation; an aggressive agent will seek personal achievement through learning and without cooperation; passive agents will do what they are told and execute tasks with the knowledge they possess and without learning.

Agents apply their knowledge to execute tasks. Their performance, cooperation and learning behaviour are influenced by their motivation, which in turn is influenced by the interaction with other agents, their own performance and the performance of the group.

3.3 The Social Model

The Social model provides the environment where agents interact and it is structured with three levels: individual, group and organisation [8]. Figure 2 shows the social model and how the agents interact with tasks.

The Agent model constitutes the individual level of the Social model. Agents are grouped in groups, or teams, and these form the organisation. The organisation task is assigned to the organisation and then distributed to teams and individuals.

Organisations and groups have their own roles and responsibilities; a repository of knowledge whether formal (documents) or informal (distributed amongst the individuals), structure and culture. These attributes determine how communication occurs and the dynamic behaviour of the group and the organisation.

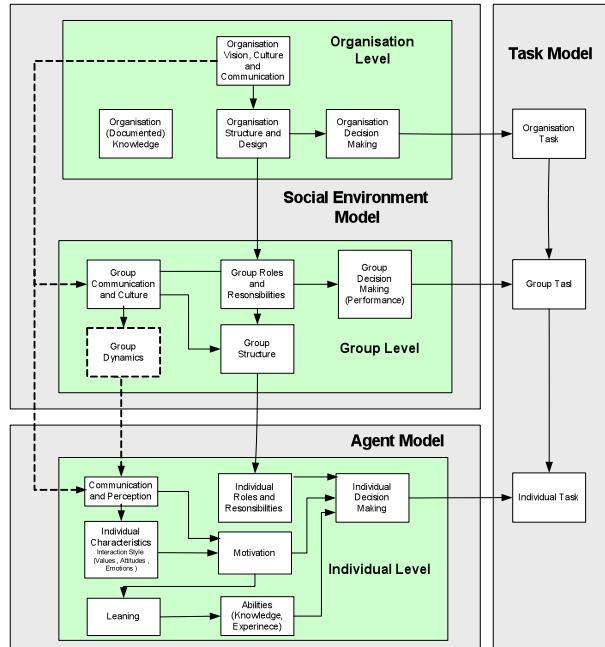


Figure 2: The Social Model.

4. THE SIMULATION

Agent-based simulation was used to implement the ACTS model. Agent-based simulation takes advantage of the processing power of computers to apply object oriented and multi-tasking techniques to simulate collections of autonomous decision-making entities called agents. Agents are software objects instantiated from specific classes that determine the behaviour of the

agent. Agents are scheduled to execute their tasks and interact with other agents in accordance with their roles, knowledge, experience and interaction styles.

Repast 3.1, the Recursive Porus Agent Simulation Toolkit [7], a Java based toolkit running on Windows environment, was used to produce the simulation.

4.1 Agent Simulation

The Agent simulates the individual, or person, that carries out a task. Each agent has a role (Team Leader or Team Member), experience (Super, Senior or Junior) and an interaction style (Constructive, Aggressive or Passive).

The “experience” attribute simulates the agent knowledge. An agent with Super experience possess all knowledge required to execute the task, represented by the normalised value of 1.0. Senior agents have less knowledge than Super and more than Junior. Senior agents have a nominal knowledge of 0.8 while Junior agents have 0.5.

Each agent holds a decision capability which is influenced by a desire to maximise the agent’s perception of the quality of the solution and its acceptance by the group. The agent has to decide when to spend time learning, asking for help or helping other agents (i.e. maximise the quality of the solution) and when to do what it is told to do to meet the imposed deadlines, knowing that the quality of the solution could be compromised due to its own lack of knowledge (i.e. maximise the acceptance of the solution).

Agents have the capacity to learn when they detect that their knowledge is not sufficient to execute the task. Learning can happen through self learning or cooperation, which consumes effort and does not progress the task assigned to the agent.

Constructive agents have a higher probability to ask for help from other agents and give help when requested. Aggressive agents have a lower probability to cooperate than for self learning. Passive agents have the lowest probability to learn or cooperate.

Team Leader agents behave in accordance with their interaction style. Constructive agents have a high probability to praise their team members when they are performing well and to offer help when they are not. This kind of interaction motivates the team members and this will increase their performance. Aggressive agents behave more aggressively and more often reprimand their team members when they are late and do not offer help. This behaviour de-motivates the agents in the team that will then perform even worse. Passive Team Leaders have the lowest probability to praise or reprimand other agents.

4.2 Task Simulation

The task simulation is based on the specification of a hypothetical naval helicopter, inspired by the acquisition of the Super Seasprite, SH-2G(A) helicopter by the Royal Australian Navy. Products are represented as vectors in the acquisition space that are transformed by transformations matrices [6].

The task comprises of six products and five transformations. There are enabling products, used during the development process, and operational products delivered with the final solution. The enabling

products are: User Need describing six operational roles to be performed by the aircraft (P_1); Operational Concept Document describing the 12 capabilities required to meet the need (P_2); Functional Performance Specification of 26 equipment and functions (P_3); System Specification and Design for each of the 26 equipment and functions, their interfaces and HMI (P_4); and the Software Specification and Design for 26 software components (P_5). The operational products are the Software Products corresponding to 26 software components (P_6). The transformations are: User Analysis (T_1), Operational Analysis (T_2), System Analysis and Design (T_3), Software Analysis and Design (T_4), Software Development (T_5). The development of these products are simulated applying equations (1) and (2) where $n = 6$ and comprising of 736 individual tasks.

4.3 Social Environment Simulation

The simulation focussed on individual and group levels. The simulation of the organisational is an extension of the group level and will be addressed in future work.

The group will be simulated by a team of agents working on a common task. Teams have team members and a team leader, while organisations have teams and managers.

The task of a team leader agent is to assign tasks to team members and to monitor the progress of these tasks. Team member agents execute the tasks assigned to them in accordance with their experience, interaction style and motivation. Team leader agents direct and attempt to control team members also in accordance with their experience, interaction style and motivation.

The interaction style of the agents determines whether they will learn and cooperate. The overall performance of teams is perceived by the agents and influences their motivation, which in turn influences their ability to apply and share the knowledge they possess, and acquire new knowledge.

4.4 Test Scenarios

The test scenarios were designed to assess the impact of knowledge and interaction styles in the time required to complete the acquisition, from the definition of the need to the implementation of software products. The transformation is complete when the effectiveness of the output product reaches its nominal value of 1.0.

Each transformation is performed by a team of agents lead by a team leader agent. There are five teams, each executing a transformation T_i . The number of agents was arbitrarily defined and is kept constant for each team.

The point of reference is the performance of agents with Super experience and Passive interaction style. The time taken to execute the tasks to completion when all teams are formed by Super Passive agents is assigned the normalised value of 1.0.

Each test scenario executes the same tasks defined by the transformation matrices T_i . The number of agents in each team is not changed, but the experience and interaction style is varied. The combination of experience and interaction styles gives 36 scenarios. There are homogeneous scenarios, where all the agents in the team have the same type of attributes (varied within a uniform distribution), such as

Senior/Aggressive, Junior/Constructive or Super/Passive; and heterogeneous scenarios, where the agents are created with two types of attributes with the same proportions, such as Super-Senior/Aggressive-Constructive or Senior-Junior/Constructive-Passive.

4.5 Results

Figure 3 shows the normalised duration for the 36 scenarios. When all agents possess super knowledge, learning and cooperation is not required to execute the task; their work is always productive and they are not influenced by the behaviour of other agents or their own. Super agents represent the ideal case with the lowest duration and effort (since the number of agents is constant) required to execute the task.

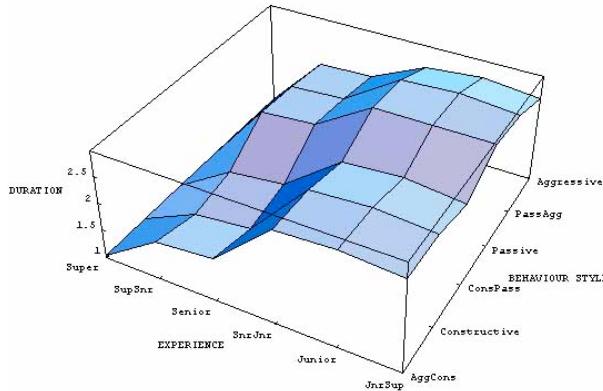


Figure 3: Duration, Experience and Interaction Styles.

Figure 3 shows an example of non-linear effect on the duration of tasks caused by available experience, knowledge and interaction styles of people executing the tasks, which makes accurate estimation and planning extremely difficult. Constructive teams perform better than teams with aggressive or passive behaviour, and teams with half of the required experience would take three times longer to execute the task.

Homogeneous teams of junior agents have the lowest performance, requiring more time to execute the task due to the fact that they have to spend effort learning. Interaction styles have no major influence in the team performance because in a homogeneous team of inexperienced agents, there are no other agents with more experience and knowledge who could help the less experienced junior agents.

Teams with senior agents perform better and are more influenced by interaction styles. Senior agents have a lower gap between their knowledge and what is required, a difference that can be acquired through self learning and cooperation with other senior agents. Heterogeneous teams with senior, junior and super (expert) agents are a closest representation of real projects, and the simulation confirms that interaction styles can influence the performance of teams and the outcomes of the acquisition as whole.

Simulation results show that team leader's interaction styles impact on team's performance. Figure 4 shows the interaction styles and motivation of the five teams. Teams 2 and 4 have aggressive team leaders, teams 1 and 5 have constructive team leaders and team 3 has a passive team leader. Aggressive team leaders are likely to make their teams become passive, while constructive

team leaders are likely to influence their teams to become constructive (Team 1) or to be even more constructive (Team 5), leading to better performance. Passive team leaders will transmit their behaviour to the team.

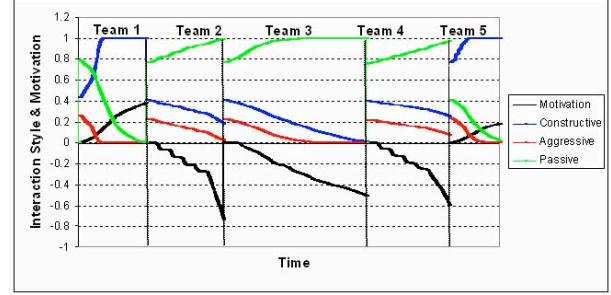


Figure 4: Influence of Team Leader's Interaction Styles and Motivation.

The results presented on Figure 3 allowed the tasks to proceed to completion but often this is not the case for real projects. Figure 5 shows the product effectiveness when team leaders stop tasks that have gone over budget before their completion, causing distortions that propagate to the subsequent transformations.

Pressures on budget and schedule can interrupt tasks before completion, producing a non-linear effect on the effectiveness of the product. In the test scenario of Figure 5, Transformations 1 (User Specification) and 5 (Software Development) have been completed, while Transformations 2 (System Specification), 3 (System Design) and 4 (Software Specification and Design) have not. Tasks completed without distortions produce output products with the nominal effectiveness represented by 1.0, while tasks interrupted before completion produce products with lower effectiveness.

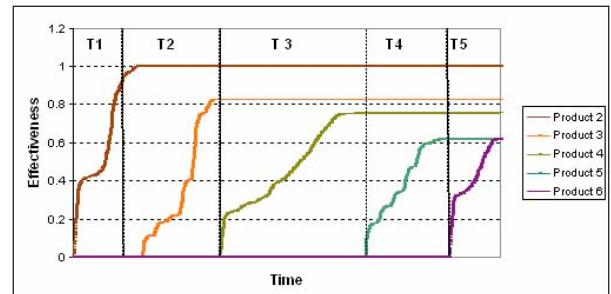


Figure 5: Influence of task interruption on Product Effectiveness.

Products with less than their nominal effectiveness are perceived as with lower quality, containing defects whose detection and correction extend schedules and increase costs.

5. CONCLUSION

The difficulty in succeeding in the acquisition of software intensive systems can be associated with the behaviour of a socio-technical complex adaptive system, where people with less than ideal knowledge and interaction styles interact to achieve their goals.

The results from a hypothetical acquisition simulation demonstrated that knowledge and interaction styles impact on group performance and project outcomes, and how non-linear effects on project duration and product effectiveness arise from the dynamic interaction of people. Realistic results could be obtained by feeding

the simulation with real data describing tasks and the knowledge required to execute them, together with the interaction styles and knowledge profiles of the people involved in the acquisition and the development processes. As this data is not always available or reliable, simulation is a useful tool to hypothesise on “what if” scenarios and evaluate alternatives with better chances of success. Simulation is a powerful aid in comprehending the complexity of software intensive acquisitions and agent-based simulation can be used as a learning tool to support the exploration of socio-organisational designs with better chances of success.

Future work will expand the proposed model and simulation to the organisation level and will investigate the effects of distributed knowledge and interaction styles as drivers of cooperation between organisations, required to detect and eliminate defects that expand between multiple domains and areas of knowledge. Future work will also investigate the effects of the stakeholders’ incentives on their motivation and the project outcomes.

REFERENCES

1. Carley, K. M., Prietulla, M. 1994, “ACTS Theory: Extending the model of bounded rationality”, In Carley, K. M. and M. J. Prietula (Eds.), *Computational Organization Theory*. Hillsdale, NJ: Lawrence Erlbaum Associates.
2. Cooke, R. A., Szumal, J. L. 1994, “The Impact of Group Interaction Styles on Problem-Solving Effectiveness”, *The Journal of Applied Behavioral Science*, 1994, vol. 30, no. 4, Dec 1994, pp. 415-437.
3. Hitchins, D. 1992, “Putting Systems to Work”, John Wiley & Sons, London, England.
4. Hoffman, L. R. 1979, “Applying Experimental Research on Group Problem Solving to Organizations”, *The Journal Of Applied Behavioral Science*, vol. 15, no. 3, July 1979, pp. 375-391.
5. Holland, J. 1995, “Hidden order: how adaptation builds complexity”, Addison-Wesley Publishing Company, USA.
6. Peculis, R., Rogers, D., Campbell, P., 2007, “A Task Model of Software Intensive Acquisitions: An Integrated Tactical Avionics System Case Study”, *12th Australian International Aerospace Congress*, Melbourne, Australia.
7. Repast, 2003, “Recursive Porus Agent Simulation Toolkit Overview”, <http://repast.sourceforge.net>.
8. Robbins, S. P. 2003, *Organizational Behavior*, Tenth Edition, Prentice Hall, Upper Saddle River, NJ.
9. Saaty T. L. 2000, “Fundamental of Decision Making and Priority Theory with the Analytic Hierarchy Process”, *Vol. VI of AHP Series*, RWS Publications, Pittsburgh, PA.
10. Stacey, R. D. 1999, “Complexity and creativity in organizations”, Berrett-Koehler Publishers, San Francisco, CA.

Appendix D: Refereed Paper 3

Title

‘An Agent-Based Framework for Simulating Socio-Organisational Design of Large Projects’

Authors

Ricardo Peculis, Derek Rogers and Peter Campbell

Source

Refereed paper presented at the Simulation Technical Conference, SimTecT 2008, held in Melbourne, VIC, Australia, from 12 to 15 May 2008.

Extended Abstract

The success of large projects depends on how people associated in teams and organisations perform their tasks and cooperate to achieve a shared goal. Large projects are complex social systems where people interact with each other and with artifacts. Socio-organisational design of complex projects involves defining organisational structures that delineate the interaction topology; determining required resources in the form of people’s abilities, roles and responsibilities; and establishing contracts and statements of work that will drive measures of success. In addition, project management is dedicated to manage the execution of the project against plans that define tasks, resources, sequence of events and expected results. Success thus depends on an effective social-organisational design.

Predicting the behaviour of a socio-organisational design of large projects is difficult, and the effectiveness of the design is likely to be validated only when put in practice. Interaction between intelligent actors produces adaptation and emergent behaviour that can move the project towards or away from its objectives. Managers and decision makers need tools to assess the effectiveness and efficiency of large projects before the project starts. Although such tools do not yet exist, simulation offers a way to explore the likely behaviour of social systems. Agent-based models, in particular, can be applied to the simulation of social systems and have been shown to be appropriate to explore

human behaviour. Simulations produced from agent-based models can be applied to aid the socio-organisational design of complex projects.

This paper proposes an agent-based framework to simulate large projects, in particular those that involve the development of complex technical systems, such as software-intensive projects and acquisitions. The framework expands BASP (Behaviour/Action Simulation Platform) framework to adapt to Axelrod's framework for 'harnessing complexity' and the ACTS (Agent, Cognition, Task and Social environment) theory. Like BASP, the framework comprises of *agents* and *connections*. *Agents* are then expanded to *actors*, that are active agents and represent people, and *artefact*⁴⁸, which are passive agents and represent physical and functional objects that are used, modified and created by *actors*. *Actors* have cognition (knowledge and perceptions) and behaviour that result in action. *Artefacts*, however, do not have cognition or behaviour. *Connections* are the way to form teams and organisations, assigning tasks to *actors* and establishing dependencies between *artefacts*. A *connection* between *actors* establishes an *interaction*; a *connection* between *actors* and *artefacts* defines a *task*; a connection between *artefacts* creates a *dependency*. The activation of *connections* determines the sequence of events and is the mechanism that creates feedback loops in the social system.

The proposed framework is then applied to a simulation using Repast, a Java based agent-based toolkit, to demonstrate that the proposed framework is suitable to investigate feedback loops in the social system, and how these feedbacks affect schedule, cost and quality of final products. The paper concludes discussing how agent-based simulations using the proposed framework can help managers and decision-makers to explore socio-organisational designs of large projects with better chances of success and avoid designs that would have higher risk of failure.

⁴⁸ The original paper adopted the alternative spelling 'artifact'.

An Agent-Based Framework for Simulating Socio-Organisational Design of Large Projects

Ricardo Peculis¹; Derek Rogers²; Peter Campbell³

¹Computer Sciences Corporation; SEEC University of South Australia,

ricardo.peculis@postgrads.unisa.edu.au

²SEEC University of South Australia, *derek.rogers@unisa.edu.au*

³SEEC University of South Australia, *peter.campbell@unisa.edu.au*

Abstract. The success of large projects depends on how people associated in teams and organisations perform their tasks and cooperate to achieve a shared goal. Success depends on an effective socio-organisational design that defines organisational structures; human resources in the form of people's abilities, roles and responsibilities; and establishing processes and contracts that will drive measures of success. Predicting the behaviour of large projects is difficult and the performance of the socio-organisational design is likely to be validated only when it is put in to practice. Simulation offers a way to explore the likely behaviour of social systems without the drawback of negative effects on real situations. Agent-based models, in particular, can be applied to the simulation of organisations. This paper proposes an agent-based framework to simulate large projects, in particular those that involve the development of complex technical systems. The proposed framework is then applied to a simulation using Repast, an agent-based toolkit, to demonstrate how the framework can be applied to investigate feedback loops in the social system, and how these feedbacks affect the schedule, cost and quality of final products. The paper concludes discussing how agent-based models and simulation using this framework can help managers and decision-makers to acquire a better understanding of social systems, to explore socio-organisational designs of large projects with better chances of success and avoiding designs that would have higher risk of failure.

1. INTRODUCTION

The success of large projects depends on how people associated in teams and organisations perform their tasks and cooperate to achieve a shared goal. Large projects are complex social systems where people interact with each other and with physical and conceptual objects that are created and modified in the course of the project. Success thus depends on an effective socio-organisational design.

Predicting the behaviour of such a design is difficult and the performance of the design is likely to only be validated when it is put in to practice. Interaction between intelligent actors produces adaptation and emergent behaviour that can create desirable and undesirable results. Managers and decision makers need to develop a better understanding of the behaviour of social systems and be supported by tools to assess the likely performance of large projects before the project starts. These tools should help to identify what socio-organisational design would have better chances of success at given conditions.

Computer simulation offers a way to explore the likely behaviour of social systems [4] and the impact of management in the life of organisations [11]. Agent-based models, in particular, can be applied to the simulation of social systems and have been shown to be appropriate to explore human behaviour [3]. Simulations produced from agent-based models can be applied to aid the socio-organisational design of complex projects [6].

This paper proposes an agent-based framework to simulate large projects, in particular those that involve the development of complex technical systems, such as software intensive projects and acquisitions. The framework expands BASP (Behaviour/Action Simulation Platform) framework [10] to adapt to Axelrod's framework for "harnessing complexity" [1] and the ACTS (Agent, Cognition, Task and Social

environment) theory [2]. Like BASP, the framework comprises of *agents* and *connections*. *Agents* are then expanded to *actors* and *artifacts*. *Actors* are active agents and represent people, while *artifacts* are physical and conceptual objects created and modified by the *actors*. *Connections* are the way to form teams and organisations, assigning tasks to *actors* and establishing dependencies between *artifacts*. A *connection* between *actors* establishes an *interaction*; a *connection* between *actors* and *artifacts* defines a *task*; a *connection* between *artifacts* establishes a *dependency*. An important aspect of *connections* is their activation, which determines when an *interaction*, *task* or *dependency* occurs. The activation of *connections* determines the sequence of events and is the mechanism that can be used to simulate feedback loops in the social system.

The proposed framework is then applied to a simulation using Repast [9], an agent-based toolkit, to demonstrate that the framework is suitable to simulate large projects; to investigate feedback loops in the social system; and to understand how these feedbacks affect the schedule, cost and quality of final products. The paper concludes discussing how agent-based simulations using the proposed framework can help managers and decision-makers to acquire a better understanding of social systems, to explore socio-organisational designs of large projects with better chances of success and avoiding designs that would have higher risk of failure.

2. SOCIO-ORGANISATIONAL DESIGN

Socio-organisational design of complex projects involves defining organisational structures that delineate the interaction of people, teams and organisations; determining required resources in the form of people's abilities, roles and responsibilities; and establishing processes, contracts and statements of work that will drive measures of success.

The performance of social systems depends on how effective and efficient the system is in achieving its objectives. Effectiveness and efficiency are intrinsic characteristics of the system itself and how the system responds to external influences [8]. The performance of large projects depend not only on organisational structures, contracts and processes but it is also largely influenced by people's experience, knowledge and behaviour [6] [7]. These are difficult to assess and there is usually no reliable data available to effectively support project planning and management.

Large projects are complex social systems subject to adaptive and emergent behaviour that can move the project towards or away from its objectives. The socio-organisational design should address these aspects of social behaviour and create structures and conditions that would make the system flexible and robust to cope with unforeseen situations without compromising business objectives.

Strategic and business needs impose tight schedules, limited budgets and demanding quality factors, on top of pressure for high efficiency that would reduce costs, increase value for money and profit. The balance between efficiency and the flexibility that the system needs to handle what has not been accounted for is a fine line that managers have to deal with [8]. In practice, there is little room for error when designing the socio-organisational systems of large projects.

After the social and organisational systems have been established, project management is dedicated to manage the execution of the project against plans that define tasks, resources, sequence of events and expected results. Project management is highly dependent and constrained by the project's socio-organisational design. Success thus depends on an effective social-organisational design.

3. SIMULATING SOCIO-ORGANISATIONS

Computer simulation offers a way to explore the likely behaviour of social systems and agent-based models, in particular, can be applied to explore human behaviour in social systems [3][4].

In accordance with the ACTS¹ theory [2], organisations are collections of intelligent agents cognitively restricted, task oriented and socially situated. In the proposed framework, people are oriented to apply their knowledge, skills and perceptions of the situation in a social environment to a problem-solving group activity, comprising of expressing the need, formulating the problem, establishing constraints and finding and implementing a solution that satisfactorily meets the need.

BASP² [10] supports *agents* and *connections*, which have *variables* and *aspects*. The novel aspect of BASP is the separation between "behaviour" and "action". *Variables* and *aspects* determine the "behaviour" and "action" of agents. According to BASP, "behaviour" establishes an "intention" of "action". Action is behaviour externalised. BASP defines *connection* as a connection between two agents and a *triple* as connections between three agents.

Socio-organisations are complex adaptive systems and Axelrod's framework for "harnessing complexity" [1] can therefore be applied for simulating socio-

organisations. Axelrod's framework is based on three main components: variation, interaction and selection. In social systems variation, or variety, is the diversity presented in the system in the form of human resources that provide knowledge and experience to perform tasks that should move the system towards its goals. Interaction is what connects people with people and with the objects they manipulate. Selection is the process that determines strategies that are good and bad and consequently the ones that should be adopted and discarded. Axelrod's framework introduces the concept of "interaction activation" that establishes timing and determines the sequence of events.

4. PROPOSED FRAMEWORK

The proposed framework, shown on Figure 1, expands BASP to adapt to Axelrod's framework and the ACTS theory. Like BASP, the framework comprises of *agents* and *connections*. *Agents* are then expanded to *actors* and *artifacts*; *connections* are expanded into *interactions*, *tasks* and *dependencies*.

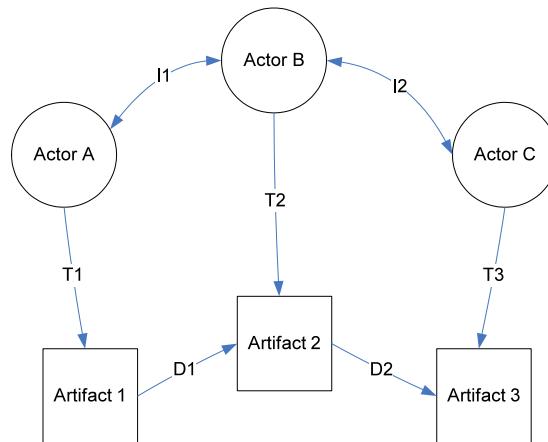


Figure 1: Proposed Agent-Based Framework.

4.1 Actors and Artifacts

Actors are active agents and represent people. *Actors* have attributes that represent the *actor's* cognition through their knowledge, emotions and motivation. These attributes determine the *actor's* behaviour which results in actions. In the course of the simulation the *actor's* attributes may change, consequently changing its behaviour and actions.

Artifacts are passive agents and represent physical and conceptual objects that are manipulated, modified and created by *actors*. *Artifacts*, however, do not have cognition or behaviour. *Artifacts* are modelled with attributes that represent the *artifact's* ideal and actual values. The difference between the ideal and actual value corresponds to the distortion or error present in the actual *artifact*.

4.2 Interactions, Dependencies and Tasks

Actors interact with other *actors* and with *artifacts* through specific *connections*. *Connections* are the way to form teams and organisations, assigning *tasks* to *actors* and establishing *dependencies* between *artifacts*. A *connection* between *actors* establishes an *interaction*; a *connection* between *actors* and *artifacts* defines a *task*; a *connection* between *artifacts* establishes a *dependency*.

¹ Agent, Cognition, Task and Social environment.

² Behaviour/Action Simulation Platform.

Interactions can happen formally, as part of the structure established by the socio-organisational design, or informally, through friendships and acquaintances. Either way, *interactions* influence the *actor's* behaviour and actions to a certain degree, depending on the *actors* and the *interaction* itself. The *interaction* defines the relationship between the *actors*, being of authority (superior/subordinate), peer or acquaintance.

Dependencies establish a dependency factor between *artifacts* and are the basis to define the *tasks* that the *actors* will perform. For products, as a collection of *artifacts*, to be effective and satisfy the need the dependency factors have to be maintained when the *actors* execute the *tasks* that are assigned to them.

Tasks define what the *actors* must do to create and modify *artifacts*. The framework proposes three types of tasks: *transformation*, *discovery* and *rework*. A *transformation task* creates *artifacts* given the input *artifact* and the transformation factor that represents the work the *actor* must perform. A *discovery task* is created and assigned to an *actor* for discovering distortions on *artifacts* already created. *Rework tasks* are in fact *transformation tasks* that modify *artifacts* to correct distortions discovered by *discovery tasks*. So that a *task* is performed without distortions the *actor* must possess the nominal level of knowledge required. *Tasks* also have a nominal level of effort that must be applied by the *actor*. If less than the nominal effort is applied by the *actor*, the *task* will be performed with distortions.

An important aspect of *connections* is their activation, which determines when an *interaction*, *task* or *dependency* occurs. The activation of *connections* determines the sequence of events and is the mechanism that creates feedback loops in the social system.

4.3 Activation of Connections and Feedback Loops

Connections have to be established and activated. A *connection* between an *actor* and an *artifact* is established when a *task* is created and it is activated when the *task* is assigned to an *actor*. *Connections* between *actors* are established when the *interaction* between the *actors* is created and it is activated when teams, organisations and acquaintances are established. *Dependencies*, the *connection* between *artifacts*, are always active; however, as *artifacts* do not have behaviour the effect of *dependencies* will occur when *actors* execute *tasks*.

The activation of *connections* can cause feedback loops in the system that impact on *artifacts* and *actors*. Figure 2 shows how feedback loops can occur. *Discovery tasks* create *rework tasks* that modify *artifacts* that through their *dependencies* impact on other *artifacts*, causing more distortions and rework.

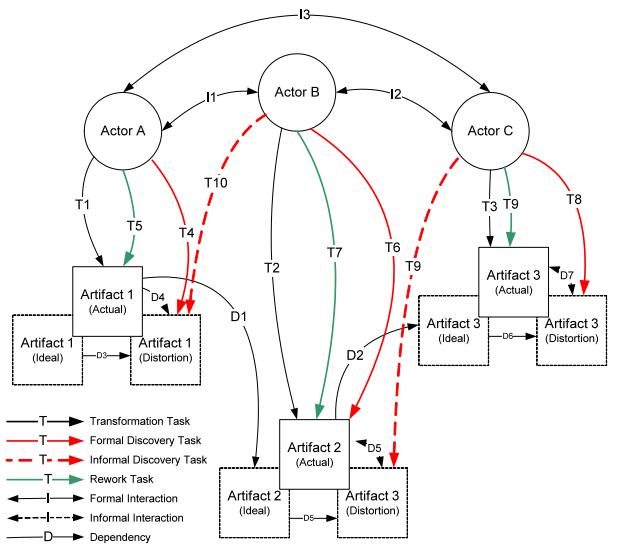


Figure 2: Feedback Loops.

Formal *discovery* and *rework tasks* included in review and test activities do not impact on the next product because the next product does not yet exist, and the rework occurs before the input to the next phase is baselined. Informal *discovery tasks* happen after the product is baselined when undetected errors are discovered during the next project phase and are fed back to the previous phase causing rework on input and output *artifacts* of the current project phase. It is not uncommon to detect defects on *artifacts* produced on earlier phases and reworking these *artifacts* will impact on *artifacts* that are connected to them through *dependencies*.

Feedback loops also occur on the social system and impact on *actors*. The activation of formal and informal *interactions* between *actors* influences *actors*' behaviour and actions. Cooperation is a positive form of interaction which can occur formally or informally. Cooperation increases shared knowledge and understanding, which in turn increases productivity and reduces errors and rework.

Another form of interaction that causes feedback loops in the social system is activated by the review process. When occurring on a constructive environment, reviews contribute positively for the project to achieve the desired goals. Reviews can also reveal problems that go beyond what is expected and may become aggressive. In these cases, reviews may trigger blaming and punishment, which increases management pressure, reduces group morale and productivity, which in turn move the project away from its objectives.

5. THE SIMULATION

The simulation was implemented with Repast 3.1 [9]. It aims to demonstrate the proposed agent-based framework and does not intend to be realistic or to reflect a real project. The simulation represents a software intensive project consisting of five phases:

- Phase 1: Operational Concept Definition,
- Phase 2: Functional Performance Specification,
- Phase 3: System Analysis and Design,
- Phase 4: Software Analysis and Design, and

- Phase 5: Software Development.

Each phase produces a product that will be input for the next phase. Distortions in one phase are propagated to the next [6]. The effectiveness of the project is determined by how well the final product highly dependent on software meets the needs that were the input for Phase 1.

5.1 Actors and Interactions

The simulation adopts the agent and social models presented in reference [6] to implement *actors* and *interactions*. *Actors* have their own knowledge, motivation, interaction style, role and responsibilities. The ideal *actor* possesses perfect knowledge and executes the *tasks* assigned to it without distortions. A more realistic *actor* has less than ideal knowledge and introduces distortions on *tasks* that it executes.

The *actor's* interaction style³ determines its willingness to learn and cooperate with other *actors* [6]. Team performance and interaction with other *actors* influence the *actor's* motivation and its productivity.

5.2 Artifacts and Dependencies

The simulation adopts a vectorial task model that represents products and tasks within the acquisition space [5] [6]⁴. Products are modelled as vectors in a Euclidian vector space, while tasks are transformation matrices that create other products. Each component of the product vectors becomes an *artifact*, and the transformation matrices provide the coefficients that define the dependencies between *artifacts*.

As an example, suppose that product P_1 is transformed into another product P_2 by the transformation matrix T (2×3), the dependency between P_2 and P_1 is represented by equation (1).

$$(1) P_2 = P_1 \cdot T$$

If P_1 has two *artifacts* (p_{11}, p_{12}), P_2 has three *artifacts* (p_{21}, p_{22}, p_{23}) and t_{ij} are the coefficients of T , the dependency of P_2 *artifacts* on P_1 is determined by equations (2), (3) and (4).

$$(2) p_{21} = p_{11} \cdot t_{11} + p_{12} \cdot t_{21}$$

$$(3) p_{22} = p_{11} \cdot t_{12} + p_{12} \cdot t_{22}$$

$$(4) p_{23} = p_{11} \cdot t_{13} + p_{12} \cdot t_{23}$$

Each component of the transformation matrix is a dependency that connects a product's *artifacts*, and becomes a *task* that is assigned to an *actor*.

5.3 Tasks

Tasks are connections between *actors* and *artifacts* and require a nominal level of knowledge and effort to be executed. If the *actor* assigned to the *task* possesses at least the nominal knowledge and applies the nominal

³ Interaction styles are classified as constructive, passive or aggressive. Constructive behaviour fosters learning and cooperation; an aggressive *actor* will seek personal achievement through learning and without cooperation; passive *actors* will do what they are told and execute tasks with the knowledge they possess and without learning [6].

⁴ Reference [6], "Using Simulation to Support the Design of Software Intensive Acquisitions", presented at SimTecT 2007, contains a summary of the Task Model [5] and presents the Agent and Social Models adopted in this simulation.

effort required, the *task* is executed without distortions. Otherwise, the *task* is executed with distortions that are proportional to the lack of knowledge and effort applied. The *task* may take longer if the *actor* decides to apply effort to learn and acquire the required knowledge.

Tasks can be *transformations*, *discovery* or *rework*. *Transformation tasks* transform input *artifacts* into other *artifacts*; *discovery tasks* aim to find distortions in *transformations* that have been completed; *rework* is another form of *transformation* that corrects distortions.

5.4 Development Process

The simulation implements a simple development process. A project phase starts when the previous phase has been completed and baselined. Each project phase receives an input *artifact*; executes all required *transformations* to their completion; performs a quality control activity; corrects the distortions that have been discovered; baseline the output *artifacts*; and communicates the baseline to the next project phase.

After completing a development cycle comprising of *transformation*, *discovery*, *rework* and *baselining*, the project can perform more *discovery* and *rework tasks*, at the end of which a new baseline is created and communicated to the next project phase. The next project phase will accept the new baseline at the end of a development cycle.

6. SIMULATION RESULTS

The simulated scenarios represent the ideal and a more realistic case. The simulation starts with the definition of the ideal *artifacts*, *dependencies* and *tasks*. The results from the ideal scenario are used as the reference to assess the results obtained from the realistic scenario.

6.1 Ideal Scenario

The ideal scenario takes the representation of the ideal *artifacts* and their *dependencies* and creates ideal *actors* capable of performing the *tasks* without distortions and within the allocated time. The effort, duration and effectiveness of the ideal case are normalised and represented by 1.0.

The graphic on Figure 3 shows the results of the ideal scenario. The "Effectiveness" shows the contribution of each project phase to the final product as a whole. Each phase provides equal contribution to the final product. The work progress, or "Work Done", progresses in accordance with the planned effort shown as "Work to be Done". There are no errors and there is no need for *discovery* and *rework tasks*.

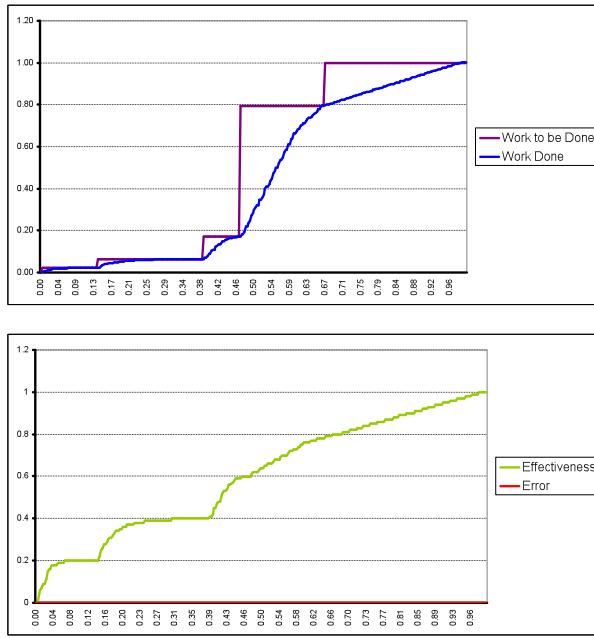


Figure 3: Ideal case.

6.2 Realistic Scenario without Learning

Figure 4 shows the results for a less than ideal case, where the same tasks are performed by teams and organisations that possess less than the required experience and knowledge. For this scenario, the simulation did not allow learning and every time a *task* is performed a constant percentage of the work produced comprises of errors that have to be discovered and corrected. Although the quantitative result is only notional, the simulation reflects of what happens on projects that struggle finding and fixing errors, which increases costs and extends schedule.

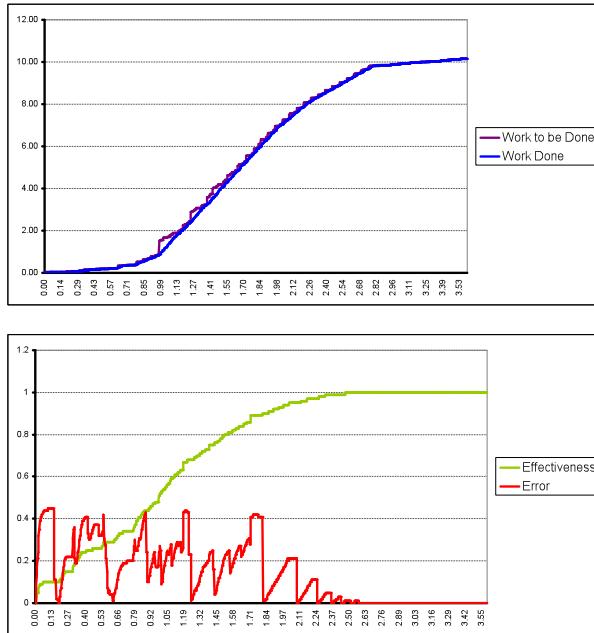


Figure 4: Less than ideal case without learning.

6.3 Realistic Scenario with Learning and Cooperation

Figure 5 shows the results for the same less than ideal scenario but with socio-organisational design that

fosters learning and cooperation. In this case the *actors* learn as they perform *tasks* and cooperate transferring knowledge and experience to each other.

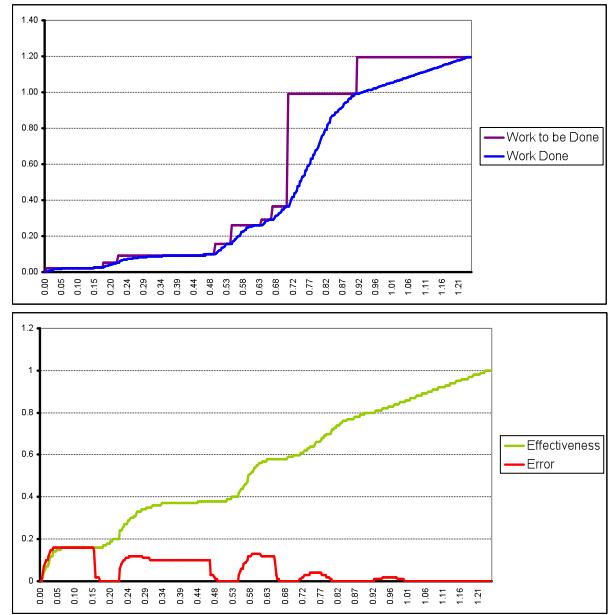


Figure 5: Less than ideal case allowing learning.

Through cooperation individual knowledge is shared and becomes collective knowledge, which is an emergent property of social systems. Every time a *task* is executed the *actors* learn and the errors produced are reduced. Learning and cooperation decrease the number of iterations, which reduces time and effort to complete the *tasks* and the project.

7. CONCLUSION

The success of large projects depends on the effective design of socio-organisations. Managers and decision makers need to develop a better understanding of the behaviour of social systems and be supported by tools to help in assessing the likely performance of the socio-organisational design before the project starts. Agent-based models and computer simulation can be used to explore the behaviour of social systems and the likelihood of success of large projects.

The proposed agent-based framework includes the basic elements that make the socio-organisational design of large projects. By modelling *actors* and *artifacts* and creating the *connections* between these agents the project can be simulated before the socio-organisational design is put in place and validated in practice. The activation of *connections* determines the sequence of events and can be used to simulate feedback loops that impact on *actors* and *artifacts*, on the social system as a whole, and influence cost, schedule and quality of the final product.

Although computer simulation provides ways to simulate social systems, modelling a real project is not simple. Obtaining reliable data to model knowledge and behaviour of the *actors*, how these attributes are influenced by *interactions* and how the *actor's* behaviour change over time is not easy. Large projects expand through multiple organisations and this kind of information, if in existence, is likely to be confidential. Organisations are also unlikely to provide information that may show they are not the best prepared for the job. If the data required to populate the model is not

available, any expectation of obtaining quantitative results from agent-based simulations is compromised.

With few exceptions, agent-based models are not intended to produce quantitative results, and are more likely to be used to explore human and social behaviour and “*what if*” scenarios. At the beginning of large projects there is an expectation that everything will happen as planned: *the customer knows what he wants; the provider is capable of delivering it; there are sufficient funds, resources and time to make it happen*. However, “*what if*” these statements are not correct? In that case, simulation offers ways to explore less than ideal scenarios and testing alternatives to compensate undesirable situations.

The benefits of simulation are not only the results the simulation provides. By investigating what is required to simulate the system a better understanding of the system itself is gained. The process of constructing a simulation of complex system can be as beneficial as, if not more, than the results that come from the simulation itself.

The proposed agent-based framework helps to simulate large projects and provides invaluable information of the building blocks that comprise their social-organisational design. By acquiring a better understanding of the social system of large projects, whether through simulation or by other means, managers and decision makers will be better prepared to develop designs with better chances of success.

REFERENCES

1. Axelrod, R., Cohen M. D. 2000, “Harnessing Complexity, Organizational Implications of a Scientific Frontier”, Basic Books, New York, NY.
2. Carley, K. M., Prietulla, M. 1994, “ACTS Theory: Extending the model of bounded rationality”, In Carley, K. M. and M. J. Prietula (Eds.), *Computational Organization Theory*. Hillsdale, NJ: Lawrence Erlbaum Associates.
3. Davis, P. K. Henninger, A., 2007, “Analysis, Analysis Practices, and Implications for Modeling and Simulation”, RAND National Defense Institute, http://www.rand.org/pubs/occasional_papers/2007/RAND_OP176.pdf.
4. Holland, J. 1995, “Hidden order: how adaptation builds complexity”, Addison-Wesley Publishing Company, USA.
5. Peculis, R., Rogers, D., Campbell, P., 2007, “A Task Model of Software Intensive Acquisitions: An Integrated Tactical Avionics System Case Study”, *12th Australian International Aerospace Conference*, Melbourne, Australia.
6. Peculis, R., Rogers, D., Campbell, P., 2007, “Using Simulation to Support the Design of Software Intensive Acquisitions”, *Simulation Technical Conference, SimTecT 2007*, Brisbane, Australia.
7. Peculis, R., Rogers, D., Campbell, P., 2007, “Embracing Knowledge And Behaviour Management To Improve Performance Of Software Intensive Projects”, *System Engineering Test and Evaluation Conference, SETE 2007*, Sydney, Australia.
8. Peculis, R., Rogers, D., Campbell, P., 2007, “Finding Better Strategies For Software Intensive Projects: The Efficiency Vs. Robustness Dilemma”, *System Engineering Test and Evaluation Conference, SETE 2007*, Sydney, Australia.
9. Repast, 2003, “Recursive Porus Agent Simulation Toolkit Overview”, <http://repast.sourceforge.net>.
10. Reynolds, W. N., Dixon, D., 2000, “A General Framework for Representing Behaviour in Agent Based Modeling”, *Complex Systems and Policy Analysis: New Tools for a New Millennium*, Arlington VA, September 2000, RAND Corporation Science & Technology Policy Institute, <http://www.leastsquares.com/papers/rand2000.pdf>.
11. Stacey, R. D. 1999, “Complexity and creativity in organizations”, Berrett-Koehler Publishers, San Francisco, CA.

Appendix E: Refereed Paper 4

Title

‘Embracing Knowledge and Behaviour Management to Improve Performance of Software Intensive Acquisitions’

Authors

Ricardo Peculis, Derek Rogers and Peter Campbell

Source

Refereed paper presented at the Systems Engineering Technical Conference, SETE 2007, held in Sydney, NSW, Australia, from 24 to 26 September 2007.

Abstract

Many reasons have been offered to explain why software projects fail. Still, software-intensive projects often present schedule delays, cost overruns and delivering products with reduced functionality. This paper argues that lack of knowledge is yet another factor that causes projects to under perform, which in turn drives undesirable social behaviour that worsens the situation. Software-intensive projects develop solutions highly dependent on software that should satisfy a need. The engineering process to develop such complex solutions comprises of a series of transformations that transform products from one domain into products in another domain, requiring knowledge pertinent to both input and output domains. Complex products call for knowledge that may not be available, is often distributed amongst people and organisations and involves learning and cooperation. The engineering of software-intensive systems, in particular, frequently adopts techniques that abstract complexity and hide details that reveal, too late, a lack of knowledge in earlier transformations. When knowledge is not applied as required, distortions propagate throughout transformations producing solutions that will not satisfy the need and require rework that increases costs and delays schedules. Software-intensive projects may fail because of lack of technical knowledge to engineer the solution, and lack of management knowledge to recognise and plan for this deficiency. When projects are not performing as expected and interests

are at risk, decisions are made within constraints that are unlikely to address the lack of knowledge, which often produces undesirable social behaviour that decreases motivation, discourages learning and cooperation and worsens the situation. The paper concludes proposing the adoption of project specific knowledge and behaviour management to improve the performance of software-intensive projects.

Embracing Knowledge And Behaviour Management To Improve Performance Of Software Intensive Projects

Ricardo Peculis

Computer Sciences Corporation;
SEEC University of South Australia,
ricardo.peculis@postgrads.unisa.edu.au

Derek Rogers

SEEC University of South Australia,
derek.rogers@unisa.edu.au

Peter Campbell

SEEC University of South Australia,
peter.campbell@unisa.edu.au

Abstract. Many reasons have been offered to explain why software projects fail. Still, software intensive projects often present schedule delays, cost overruns and delivering products with reduced functionality. This paper argues that lack of knowledge is yet another factor that causes projects to under perform, which in turn drives undesirable social behaviour that worsens the situation. Software intensive projects develop solutions highly dependent on software that should satisfy a need. The engineering process to develop such complex solutions comprises of a series of transformations that transform products from one domain into products in another domain, requiring knowledge pertinent to both input and output domains. Complex products call for knowledge that may not be available, is often distributed amongst people and organisations and involves learning and cooperation. The engineering of software intensive systems, in particular, frequently adopts techniques that abstract complexity and hide details that reveal, too late, a lack of knowledge in earlier transformations. When knowledge is not applied as required, distortions propagate throughout transformations producing solutions that will not satisfy the need and require rework that increases costs and delays schedules. Software intensive projects may fail because of lack of technical knowledge to engineer the solution, and lack of management knowledge to recognise and plan for this deficiency. When projects are not performing as expected and interests are at risk, decisions are made within constraints that are unlikely to address the lack of knowledge, which often produces undesirable social behaviour that decreases motivation, discourages learning and cooperation and worsens the situation. The paper concludes proposing the adoption of project specific knowledge and behaviour management to improve the performance of software intensive projects.

INTRODUCTION

Many reasons have been offered to explain why software projects fail. Still, software intensive projects often present schedule delays, cost overruns and the delivery of products with reduced functionality. This paper argues that lack of knowledge is yet another factor that causes projects to under perform, which in turn drives undesirable social behaviour that worsens the situation.

Software intensive projects develop solutions highly dependent on software that should satisfy a need. The engineering process to develop such complex solutions comprises of a series of transformations that transform products from one domain into products in another domain, requiring knowledge pertinent to both input and output domains. Complex products call for knowledge that may not be available, is often distributed amongst people and organisations and involves learning and cooperation. The engineering of software intensive systems, in particular, frequently adopts techniques that abstract complexity and hide details that reveal, too late, a lack of knowledge in earlier transformations. When knowledge is not applied as required, distortions propagate throughout transformations producing solutions that will not satisfy the need and require rework that increases costs and delays schedules.

Executive support, customer involvement and experienced project managers, together with clear business objectives, top the list of factors that have been reported as key contributors to success of software intensive projects (Standish, 1998, 2001). Lower in the list, but still in the top ten, are also stable requirements and competent staff. When these factors are put into context, their level of influence and importance may be

questioned. Software intensive projects are business enterprises and should meet objectives for customers and providers. The involvement of competent staff in both customer and provider organisations is fundamental to establish requirements and contractual conditions that should satisfy both parties. Executive support creates enabling conditions, while experienced project managers plan the tasks, monitor them and intervene when deviations occur. Staff and managers must possess knowledge and skills to execute their tasks.

Software intensive projects may fail because of lack of technical knowledge to engineer the solution, and lack of management knowledge to recognise and plan for this deficiency. When projects are not performing as expected and interests are at risk, decisions are made within constraints that are unlikely to address the lack of knowledge, which often produces undesirable social behaviour that decreases motivation, discourages learning and cooperation and worsens the situation.

The purpose of specific knowledge management is to identify the knowledge and skills required to engineer the solution, to ascertain what of this knowledge and skills is available and what is not; and to develop a plan to harvest distributed knowledge and to acquire the knowledge that is missing. Specific behaviour management aims to observe the social behaviour of teams and organisations, and to develop strategies to promote constructive behaviour that fosters learning and cooperation. The paper concludes by proposing the adoption of project specific knowledge and behaviour management to improve the performance of software intensive projects.

SOFTWARE INTENSIVE PROJECTS

Software intensive projects apply the process of engineering to develop products highly depend on software. As the scale¹, life cycle² and interdependency of internal components of engineering projects increase, their complexity also increases (Aslaksen, 1996). Complexity refers to the limitation of the human brain³ in the visualisation and manipulation of systems that involve a large number of non linear interactions between components and parameters. To be able to handle complex objects, the brain removes what it judges to be less important or relevant but even with such a “fuzzy picture” the brain is capable of manipulating complex situations with complex entities.

To handle large projects, partitioning is essential. From the design perspective, functional requirements are partitioned in the most logical way, while management will break up the work in a most efficient work breakdown structure. While project management deals with physical entities such as the coordination and management of the project activities, engineering management deals with functional entities (Aslaksen, 1996). Physical entities are more tangible and are less difficult to understand and communicate, while functional entities are more difficult to understand and manipulate, due to the manner in which the brain deals with complex tasks.

Engineers and managers are subject to limitations in dealing with complex entities and situations. Consequently, mental pictures created by the cognitive process are a simplified version of reality. As the brain removes what it finds less important or unnecessary, the mental picture is influenced by knowledge, emotions, beliefs and preferences. As the result, the mental picture is likely to be incomplete, distorted and biased. The accuracy of mental pictures and the way they are shared by the team is of fundamental importance for the success in achieving the common goal (Newell et al, 1990).

Experience shows that the engineering of products with a large software component is not an easy task. The challenge can be associated to the fact that for the development of software, functional products extend longer throughout the project, and are more difficult to communicate about and estimate than the hardware components. The engineering of software intensive systems, in particular, frequently adopts techniques that abstract complexity and hide details that then reveal problems at late stages of the project.

In addition to the complexity of the multitude of interacting technical components, large software intensive projects are likely to be executed by multiple teams and organisations, increasing the complexity of the project.

The Acquisition Model

The aim of the acquisition process is to acquire a solution that will satisfy a need, and software intensive acquisitions, in particular, acquire solutions that depend on engineering projects to develop products with software components. The acquisition process can be represented by a set of products and transformations

¹ Scale refers to size, cost and number of components (Aslaksen, 1996).

² Life cycle refers to environmental and temporal dependencies (Aslaksen, 1996).

³ Studies show that the brain has difficulty in manipulating entities which comprise more than seven elements (Aslaksen, 1996).

within the acquisition space, comprising of three subspaces: Situation, Problem and Solution (Peculis et al, 2007). The acquisition originates in the Situation domain, where a need exists and can be expressed by a set of products pertinent to that space. The need leads to the definition of the problem as another set of products in the Problem domain. The solution that resolves the problem comprises a set of products in the Solution domain. When the solution is applied to the Situation it should satisfy the need. Transformations consist of tasks that apply engineering processes transforming products from one domain into products in another domain. The execution of these tasks requires knowledge associated with the domain space of the input and output products. The quality of the products depends on people's knowledge and skills and their motivation to execute the task.

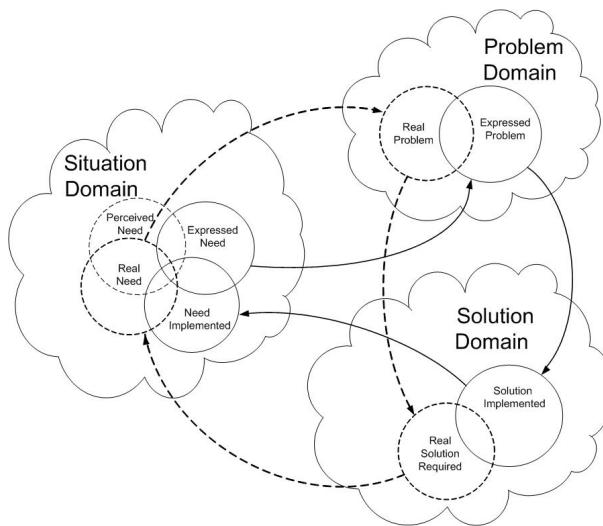


Figure 1: Transformations between Domain Spaces.

Ideally, transformations should occur without distortions. The need should be accurately expressed, the problem well defined, the solution correctly implemented to address satisfactorily the need. In reality, distortions occur because people within social organisations, limited by their own knowledge, skills, cognition, emotions, moved by personal and corporate goals and bounded by their roles in the social organisation, are responsible for the execution of transformations between and within domains. Distortions are carried from one domain to another and propagate with a “Chinese Whispers” effect (Peculis et al, 2007).

The effectiveness of the acquisition process indicates how well the acquisition achieves its goals and efficiency is measured by how much waste is produced in the process. From Figure 1, effectiveness indicates how close the implemented solution came to fulfilling the real need, i.e. the unbroken line circles coinciding with the dotted circles, and efficiency shows how much effort, in the form of enabling tasks and rework, is required to bring the implemented solution to meet the real need. Lack of knowledge and skills available, or lack of applying knowledge and skills available, are some of the causes of low effectiveness and efficiency in software-intensive acquisitions.

THE INFLUENCE OF KNOWLEDGE

The need for knowledgeable, talented and motivated people to develop software products is undeniable (DAF, 2000) (SEI, 2001). Knowledge is not only of

fundamental importance to providers that formulate the problem and engineer the solution, but also to customers in their ability to communicate the need (DAF, 2000). Lack of management knowledge also has been reported as one of the causes of failure of software projects (Perkins, 2006). Lack of knowledge is thus another factor that causes projects to under perform.

Distributed and Incomplete Knowledge

No single individual or organisation possesses all knowledge, nor has access to all information required to specify, design and implement complex software intensive systems. Individuals and organisations are likely to be more knowledgeable on subjects that are pertinent to the domain where they operate. Distributed knowledge often leads to incomplete information and creates different perceptions and mental models. Distributed and unshared knowledge turn teams and organisations away from common understanding, common goals, and affects the system effectiveness and efficiency.

The effects of incomplete information and distributed knowledge combined with less than ideal skills create distortions that propagate through specification and design documents, resulting into incomplete or even wrong information when they reach the software domain. Problems that are not addressed and resolved in their domain of origin will propagate through the acquisition space to other subspaces where the required knowledge to address the problems is inadequate or non-existent. These distortions become defects that individuals, teams and organisations will try to back loop and correct, and are drivers for management intervention.

The larger the number of organisations in the system, the greater these effects of distributed knowledge become. The acquisition process, and consequently the software intensive project, ought to impel the acquisition organisation, as a collection of several organisations, to harvest and apply collective knowledge. Collective knowledge is an emergent property, whose effectiveness should be maximised by the organisational design. The distributed nature of knowledge in software intensive acquisitions is a fact that must be inserted into the organisational design.

THE INFLUENCE OF BEHAVIOUR

Although knowledgeable people are great assets for the success of software intensive projects, that may not be enough. People need to be motivated to do the job, which often requires doing activities that are not specifically included as part of the original task. Depending on the complexity of the task that may include acquiring new skills, learning new subjects, cooperating, teaching and helping others. Motivated people perform beyond their formal duties.

Depending on their behaviour⁴ style, people respond differently to situations that are presented to them, some producing better results than others. Managers, for example, when facing a late project, may force the team to work harder, or may offer help to address what is causing the delay. Certainly, these actions would cause different reactions and a different response from each individual team member. While constructive attitudes are likely to motivate the team, aggressive behaviour tends to decrease motivation.

⁴ Behaviour refers to how people react in response to a stimulus.

Problem-Solving Groups

Software intensive projects can be considered as problem-solving groups. Problem-solving groups have two principal objectives: (1) quality of results, that requires the maximum utilisation of resources brought by each individual member of the group; and (2) acceptance of the solution, that implies a high level of motivation for carrying out the group's decision (Hoffman, 1979). These objectives create pressure on each member of the group to decide between bringing their unique, and possibly controversial, contribution to the problem-solving task, or to conform to group's decisions (Hoffman, 1979).

The effectiveness of problem-solving groups depends on the group's interaction style, which can be constructive, aggressive or passive (Cooke & Szumal, 1994). Groups with a constructive style tend to maximise the contribution of each group member, fostering cooperation between the group members to achieve both individual and group objectives. Members of groups with a passive style behave in ways that promote their security and acceptance, avoiding conflict, limiting information sharing, and allowing themselves to become dominated by the group. In the aggressive group style, members of the group will behave to promote their status and position to fulfil their own needs; members also tend to demonstrate their competence doing things perfectly, without seeking help or helping others and often losing sight of the group's goals.

Across problem-solving groups, solution quality increases when the group shows a constructive interactive style and decreases with passive interaction style; and the acceptance of the solution increases with constructive interaction style and decreases with both passive and aggressive interaction styles; aggressive interaction has been shown to be unrelated to the quality of the solution (Cooke & Szumal, 1994).

The Influence of Stakeholders

Another important aspect to be observed is the influence that stakeholders have over the project. Senior management, whether from customer or provider organisations, have high influence over the project and are capable of support or ending the project. Senior managers are influenced by other stakeholders that are external to the project, like politicians, the media, marketing personnel, competitors, etc. These influences can also support or end the project. Internal stakeholders, such as project managers, senior engineers and natural leaders can also influence the success or failure of the project.

The increase in the level of abstraction and extended design of software intensive projects expands physical expenditure (e.g. cost, schedule, resources) which delays the physical or tangible results. When projects are not performing as expected and interests are at risk, the nature of engineering and management activities, i.e. functional vs. physical, creates tensions between engineering management and project management (Aslaksen, 1996). Management decisions are then made within constraints that are unlikely to address the lack of knowledge, which often produces undesirable social behaviour that decreases motivation, discourages learning and cooperation and worsens the situation. Experience also shows that in the end, political and business factors are favourable to the sort of project management that holds the control over the project and

determines what to do and when. As a result, the design process is compromised, the quality of the solution decreases and originates rework, which in turn increases cost and delays the schedule.

KNOWLEDGE MANAGEMENT

Knowledge management is the process of creating, disseminating and applying knowledge, often represented graphically as a pyramid, shown on Figure 2. The first layer of knowledge management comprises of gathering *data*, which after being interpreted and put into context becomes *information*, the second layer of the process. The third layer focuses on how information is used and becomes *knowledge*. The top two layers include *understanding* and *wisdom*; the former reflects the comprehension and awareness of the consequences that the use of knowledge can bring; the latter is the result of the ability of conscientiously taking advantage of possessing and applying knowledge.

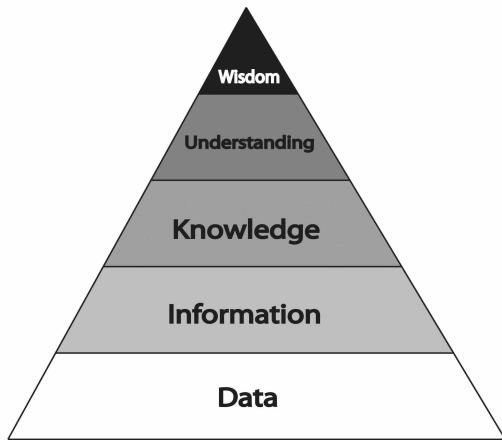


Figure 2: Knowledge Management Philosophy.

Within the context of software intensive projects, the process described by the five layers of the knowledge management pyramid is usually seen as a philosophy rarely put in practice. Although knowledge is a fundamental element for the success of software intensive projects, it is unusual to include a Knowledge Management Plan in the myriad of plans required to start a project. The adoption of project specific Knowledge Management Plan can bring benefits to software intensive projects.

Specific Knowledge Management Plan

The objective of a project specific Knowledge Management Plan (KMP) is to identify the knowledge required to execute the project, the knowledge that is

available, and the knowledge that is missing; and to create a plan to harvest existing knowledge and to acquire the knowledge that is not readily available. Through an analysis of user and system requirements, proposed system design and the technology required to execute the project, the KMP identifies the various areas of knowledge, where the existing knowledge is located, and establishes strategies to create a “collective knowledge”.

Some of the key aspects included in the KMP are methods and processes for measuring and assessing intangible assets, such as knowledge, skills and experience, and the policies for selecting organisations and staff for the project.

Software intensive projects are often executed by multiple organisations that extend from the customer to prime contractor and subcontractors. Effective results will emerge when specific knowledge management is applied and extended to all organisations involved in the software intensive acquisition as a whole.

KMP for a Naval Helicopter

The following example, based on the specification of a naval helicopter illustrates the process of identification of areas of knowledge and their dependencies (Peculis et al, 2007). The specified roles for this naval helicopter are: Surveillance (SV), Anti-Ship Warfare (ASW), Anti-Submarine Warfare (ASbW), Search & Rescue (S&R), Support Missions (SM) and Training Support (TrS). Table 1 represents the user's requirements in the form of the aircraft's roles and capabilities expressed by the Operational Concept Document (OCD). The “dots” on Table 1 show the dependencies between the aircraft's roles and capabilities, and correspond to the tasks that need to be executed to prepare the OCD that accurately expresses the customer's need. Each task corresponds to a capability placed within the context of the aircraft's roles, and requires a specific knowledge that combines two domains of knowledge: aircraft's operational roles and aircraft's capabilities. The Human/Machine Interface (HMI) for Surveillance (HMI/SV) and HMI for Anti-Submarine Warfare (HMI/ASbW) comprise two distinctive domains of knowledge. The first combines the knowledge of HMI with the knowledge of Surveillance, while the latter applies the knowledge of HMI into Anti-Submarine Weapons. Similarly, to specify Tactical Data Management (TDM) is required knowledge of TDM/SV, TDM/ASW, TDM/ASbW and TDM/TrS. Table 1 thus suggests that the preparation of the OCD for the naval helicopter of the example is required knowledge of 58 specific areas, determined by the number of “dots” on Table 1.

Aircraft Roles ►	SV	ASW	ASbW	S&R	SM	TrS
Aircraft Capabilities ▼						
HMI for crew of two (HMI)	•	•	•	•	•	•
Aircraft Monitor & Supervision (M&S)	•	•	•	•	•	•
Mission Preparation Support (MPS)	•	•	•	•	•	•
Mission Debrief Support (MDF)	•	•	•	•	•	•
Electronic Navigation (NAV)	•	•	•	•	•	•
Radio Communications (COM)	•	•	•	•	•	•
Surveillance Sensors (SVS)	•	•	•	•	•	•
Electronic Warfare (EW)	•	•	•			•
Tactical Data Management (TDM)	•	•	•			•
Tactical Navigation Mgmt and AFC (TNM)	•			•	•	•
Anti-Ship Weapons (WAS)			•			•
Anti-Sub Weapons (WASb)			•			•

Table 1: OCD: Aircraft's Roles and Capabilities

Table 2 shows the dependencies between the specification of the aircraft's system components and the specification of the aircraft's capabilities. Table 2 corresponds to what is included in the Functional Performance Specification (FPS), prepared by 155 tasks, indicated by the "dots" on Table 2. To specify the requirements for the Radar, for example, it has to be placed within the context of several capabilities (HMI, M&S, MPS, MDS, SVS and TDM), and requires knowledge of the operation of a specific Radar applied to these capabilities. The example suggests that 155 tasks, determined by the number of "dots" on Table 2, are needed to prepare the FPS. The tasks to prepare the OCD and FPS call for the application of 213 specific areas of knowledge.

The process of identification of areas of knowledge continues into the specification and design of the system and the software until the development of software components. For each of the 26 system components, specification and design documents are prepared to address the system (SSS/SSDD⁵), equipment interfaces (IRS/IDD⁶) and human interfaces (HIS/HIDD⁷). The specification and design of each system component (e.g. Radar, EW, Weapons, etc.) calls for specific knowledge of that component in the context of equipment interfaces, human interfaces and the equipment operation within the system. Thus, each system component calls for six specific areas of knowledge to develop the SSS, SSDD, IRS, IDD, HIS and HIDD, which for 26 system components adds the need for a total of 156 areas of knowledge.

The specification and design of the system and its interfaces become the input for the software specification and design (SRS/SDD⁸) for each of the 26 CSCIs⁹. Each CSCI (e.g. Radar, EW, Weapons, etc.) requires specific knowledge for the specification and design of the software, creating a need for 52 areas of knowledge, two (i.e. specification and design) for each CSCI. Finally, with a complete software specification and design, the CSCIs are coded and tested, which should require only knowledge of software development. The software specification and design that

allows software developers to do their job thus contain information derived from over 420 areas of knowledge, that were required to develop the OCD, FPS, and SSS, SSDD, IRS, IDD, HIS and HIDD for 26 systems components and CSCIs.

The KPM applies the Knowledge Management philosophy in a specific and practical way to bring the required knowledge into the project. The KMP not only identifies the areas of knowledge required to do the job but also whether the knowledge is available and where it is located. Knowledge of operational aspects of the aircraft, including roles and capabilities, are likely to be available in the Situation domain. It is unlikely for systems engineers to know everything about the Situation domain, and a close cooperation between the users and engineers is often needed. Similarly, due to the fact that equipment manuals and interface documents are often incomplete and misleading, cooperation between systems engineers and equipment manufacturers is needed to specify and design system components. The KMP includes a plan to harvest the available knowledge through workshops and working groups that gather users, engineers and experts together. The KPM also includes plans for formal training and time for self learning.

⁵ SSS: System/Subsystem Specification; SSDD: System/Subsystem Design Document.

⁶ IRS: Interface Requirements Specification; IDD: Interface Design Document.

⁷ HIS: Human Interface Specification; HIDD: Human Interface Design Document.

⁸ SRS: Software Requirements Specification; SDD: Software Design Document.

⁹ CSCI: Computer Software Configuration Item.

Capabilities ►	HMI	M&S	MPS	MDS	NAV	COM	SVS	EW	TDM	TNM	WAS	WASb
Components ▼												
Dual Data Processor	•	•	•	•	•	•	•	•	•	•	•	•
Data Entry & Display	•	•	•	•	•	•	•	•	•	•	•	•
MDLR	•	•	•	•	•	•	•	•	•	•	•	•
Aircraft Sensors	•	•	•	•	•	•						
ADC	•	•	•	•	•							
INS/GPS	•	•	•	•	•							
VOR/ILS	•	•	•	•	•							
DME	•	•	•	•	•							
RADALT	•	•	•	•	•							
LF ADF	•	•	•	•	•							
IFF	•	•	•	•	•							
Intercom	•	•	•	•	•							
HF Radio	•	•	•	•	•							
U/VHF Radio	•	•	•	•	•							
Radar	•	•	•	•	•							
FLIR	•	•	•	•	•							
ESM	•	•	•	•	•							
Radar Warning System	•	•	•	•	•							
Missile Warning System	•	•	•	•	•							
Laser Warning System	•	•	•	•	•							
CMDS	•	•	•	•	•							
Link-11	•	•	•	•	•							
AFCS	•	•	•	•	•							
Anti-Ship Missile	•	•	•	•	•							
Torpedo	•	•	•	•	•							
Depth Bomb	•	•	•	•	•							

Table 2: FPS: Aircraft's Capabilities and System Components.

BEHAVIOUR MANAGEMENT

Within the context of software intensive projects, the objective of Behaviour Management is to improve the project outcomes through the way the people and or organisations interact. Behaviour expectation driven by organisational culture can drive the level of cooperation in teams and groups; while constructive behaviour is likely to impel cooperation, non-constructive interaction styles can undermine attempts at knowledge exchange and of achieving knowledge management goals (Balthazard & Cooke, 2004). Constructive behaviour enhances the quality and acceptance of the solution produced by problem-solving groups (Cooke & Szumal, 1994). Behaviour Management thus intends to maximise constructive behaviours, promote cooperation and increase the effectiveness of Knowledge Management.

Specific Behaviour Management Plan

The objective of the proposed project specific Behaviour Management Plan (BMP) is to identify the likely behaviour within the project's social environment and create a plan to maximise the effectiveness of the organisation through stakeholder and organisational culture management.

Stakeholder management identifies people and organisations capable of influencing the project and develops strategies to create a culture of constructive behaviours. Stakeholder management attempts to ascertain stakeholder's interests and their likely behaviour, and to identify project risks that could come from stakeholders capable of swaying the project negatively. The BMP develops strategies to educate non-constructive stakeholders, managers and team members to change their interaction styles and attitudes towards more constructive behaviours. Stakeholder management is essential for effective project management (Bourne & Walker, 2003). The Stakeholder Circle (Bourne & Walker, 2006) is a stakeholder management tool to identify, prioritise and developing a stakeholder engagement strategy that

better suites the project, or business enterprise, objectives.

The aim of Organisational Culture Management is also to maximise the effectiveness of the organisation by developing an organisational culture that best suites the organisation's objectives. Organisations that deal with software intensive projects and problem-solving groups in general, tend to increase their effectiveness through constructive cultures. Behaviour management tools such as the Organisational Culture Inventory (OCI) (Cooke & Szumal, 2000) and the Life Style Inventory (LSI) (Cooke & Rousseau, 1987) use surveys to produce qualitative data about organisational culture and the behaviour profile of staff members that are capable of influencing the culture of the organisation. As the result of making people aware of their strengths and limitations, and through appropriate training, cultural change programs aim to reshape the organisation in ways to improve its effectiveness. When applied to software intensive projects, behaviour management should also be extended to all organisations involved in the software intensive acquisition to be able to produce effective results.

CONCLUSION

Experience shows that software intensive projects are prone to schedule delays, cost overruns and functional deficiencies when the product is delivered. The diversity of knowledge required, the functional and abstract nature of software products, together with human limitations to deal with large number of interconnected entities make the development of software intensive systems a challenging endeavour. As the complexity of software systems increase, the complexity of the social organisation required to engineer them also increases.

Software intensive projects are problem-solving groups, and success depends on how mental pictures are created and communicated and the way people interact and apply their knowledge. The chances of success are maximised when people behave constructively rather than aggressively or passively. The success of software

intensive projects thus depends on knowledgeable, talented and motivated people. Learning and cooperation is of fundamental importance to acquire missing knowledge and to transform distributed knowledge into collective knowledge. Collective knowledge is an emergent property that should be addressed by the organisational design of software intensive acquisitions.

Knowledge and behaviour influence each other. Lack of knowledge often leads to errors and deficiencies that require rework that delays schedules and increase costs, often causing management intervention that is unlikely to address the root cause of the problem. This kind of aggressive attitude causes other undesirable social behaviour that lowers motivation to learn and cooperate and worsens the situation.

Specific Knowledge Plan and Behaviour Management Plan concepts are proposed to address knowledge and behaviour components of software intensive projects. The former aims to identify the knowledge required and available; and to develop a plan to acquire the knowledge that is missing and to create conditions to enable sharing of distributed knowledge. The latter intends to create conditions that will foster constructive behaviours, learning and cooperation; and to avoid non-constructive behaviours that will move the project away from desirable outcomes.

In conclusion, embracing Knowledge and Behaviour Management, through realistic plans and their effective application is an important element of socio-organisational design that can improve performance and chances of success of software intensive projects. Effective results will emerge when the commitment to Knowledge and Behaviour Management is embraced by all organisations involved in the software intensive acquisition as a whole.

REFERENCES

1. Aslaksen E. W. 1996, *The Changing Nature of Engineering*, McGraw-Hill Australia, Sydney.
2. Balthazard, P. A., Cooke, R. A., 2004, Organizational Culture and Knowledge Management Success: Assessing The Behavior-Performance Continuum, *Proceedings of the 37th Hawaii International Conference on System Sciences – 2004*.
3. Bourne, L., Walker, D. H. T., 2003, "Tapping the Power Lines: A 3rd Dimension of Project Management Beyond Leading and Managing", *7th Australian International Performance Management Symposium*, Canberra, Australia.
4. Bourne, L., Walker, D. H. T, 2006, "Using a Visualising Tool to Study Stakeholder Influence", *The Project Management Journal*, Vol 37 No. 1, 2006.
5. Cooke, R. A., Rousseau, D. M., 1987, "Thinking and Behavioral Styles: Consistency Between Self-Descriptions and Description by Others", *Educational and Psychological Measurement*, 1987, 47.
6. Cooke, R. A., Szumal, J. L. 1994, "The Impact of Group Interaction Styles on Problem-Solving Effectiveness", *The Journal of Applied Behavioral Science*, 1994, vol. 30, no. 4, Dec 1994, pp. 415-437.
7. Cooke, R. A., Szumal, J. L, 2000, "Using Culture Inventory to Understand the Operating Culture of Organizations", *Handbook of Organizational Culture & Climate*, 2000, pp.147-162, Neal M. Ashkanasy, Celeste Wilderom, Mark F. Peterson, editors.
8. DAF, Department of Air Force, 2000, *Guidelines for Successful Acquisition and Management of Software-Intensive Systems*, Version 3.0, May 2000, viewed 10 Nov. 2003, <<http://web.nps.navy.mil/~menissen/mn3309/stsc-guidelines3.0/GSAMv3.pdf>>.
9. Hoffman, L. R. 1979, "Applying Experimental Research on Group Problem Solving to Organizations", *The Journal Of Applied Behavioral Science*, vol. 15, no. 3, July 1979, pp. 375-391.
10. Newell, A., Yost, G., Laird, J., Rosenbloom, P., & Altman, E. 1990, *Formulating the problem space computational model*. Paper presented at the 25th Anniversary Symposium, School of Computer Science, Carnegie Mellon University.
11. Peculis, R., Rogers, D., Campbell, P., 2007, "A Task Model of Software Intensive Acquisitions: An Integrated Tactical Avionics System Case Study", *12th Australian International Aerospace Conference*, Melbourne, Australia.
12. Perkins, T. K. 2006, "Knowledge: The Core Problem of Project Failure", *CrossTalk, The Journal of Defense Software Engineering*, Vol. 19 No. 6, June 2006.
13. SEI, Software Engineering Institute, 2001, People Capability Maturity Model, P-CMM, V2.0, *Carnegie Mellon University*, Pittsburgh, PA.
14. Standish, G 1998, 'CHAOS: A receipt for success', Standish Group, viewed 13 Nov. 2003, <http://www.standishgroup.com/sample_research/PDFpages/chaos1998.pdf>.
15. Standish, G 2001, 'Extreme Chaos', Standish Group, viewed 13 March. 2004, <http://www.standishgroup.com/sample_research/PDFpages/extreme_chaos.pdf>.

BIOGRAPHY

Ricardo Peculis - Mr Peculis is a senior systems engineer at Computer Sciences Corporation, where he has been involved with major defence projects as a senior engineer and technical manager. The challenge to succeed in software intensive projects motivated him to further investigate the causes that drive undesirable results and ways to improve the effectiveness of such projects. Mr Peculis is currently a PhD student at the Systems Engineering Evaluation Centre of the University of South Australia and his research addresses the implications of socio-organisational design into the complexity of software intensive acquisitions.

Derek Rogers - Dr Derek Rogers is an adjunct senior research fellow at the University of South Australia and Product Development Manager for BAE Systems. He focuses in the areas of systems engineering, wireless telecommunications, innovation and commercialisation. Dr Rogers has published approximately thirty papers and also holds an adjunct role at the University of Adelaide and visiting lecturer role at the University of Canterbury, New Zealand.

Peter Campbell - Professor A. Peter Campbell is currently (2004-7) working under contract with CSIRO Complex Systems Science Initiative to introduce complex system simulation tools for agricultural landscape planning and critical infrastructure analysis. Since May 2004, Professor of Systems Simulation and Modelling with the Systems Engineering and Evaluation Centre (SEEC) at the University of South Australia working on the application of complex adaptive system simulation technology to large scale

system integration projects at UniSA and development of the Centre for Defence and Industry Systems Capability(CEDISC). Recent work includes development of agent based simulations to support organizational change within the Australian Defence Organization.

Appendix F: Refereed Paper 5

Title

'Finding Better Strategies For Software Intensive Projects: The Efficiency Vs. Robustness Dilemma'

Authors

Ricardo Peculis, Derek Rogers and Peter Campbell

Source

Refereed paper presented at the Systems Engineering Technical Conference, SETE 2007, held in Sydney, NSW, Australia, from 24 to 26 September 2007.

Abstract

The causes of failure of large software-intensive projects can be attributed to the complexity of today's technical systems and to the even more complex social systems that are required to engineer them. Planning and control of complex systems is a difficult task. The greater the complexity, the higher are the chances that the system will encounter situations that have not been considered. Projects aim to be effective and efficient. Effectiveness is an indication of how well the project achieves its goals, and efficiency shows how much is wasted in the process. Complex systems need diversity to be able to adapt to unplanned conditions and achieving their goals. Diversity increases robustness of social systems, providing what is needed to achieve their goals under adverse conditions. However, diversity may create organisational capacity in excess of what is needed and reduce the system's efficiency, which possibly will conflict with business and political interests. This creates the efficiency vs. robustness dilemma. The development of Linux is presented as a case of a robust Complex Adaptive System software development, and compared with evolutionary and traditional engineering approaches. Linux was developed with no commercial interests, without any cost or schedule constraints and did not aim to be profitable. The efficiency vs. robustness of the development of Linux is then discussed within the context of what is expected of commercial software development. In conclusion, the paper argues that sacrificing

efficiency for robustness can be a sensible approach for strategic and safety critical software-intensive projects

Finding Better Strategies For Software Intensive Projects: The Efficiency Vs. Robustness Dilemma

Ricardo Peculis

Computer Sciences Corporation;
SEEC University of South Australia,
ricardo.peculis@postgrads.unisa.edu.au

Derek Rogers

SEEC University of South Australia,
derek.rogers@unisa.edu.au

Peter Campbell

SEEC University of South Australia,
peter.campbell@unisa.edu.au

Abstract. The causes of failure of large software intensive projects can be attributed to the complexity of today's technical systems and to the even more complex social systems that are required to engineer them. Planning and control of complex systems is a difficult task. The greater the complexity, the higher are the chances that the system will encounter situations that have not been considered. Projects aim to be effective and efficient. Effectiveness is an indication of how well the project achieves its goals, and efficiency shows how much is wasted in the process. Complex systems need diversity to be able to adapt to unplanned conditions and achieving their goals. Diversity increases robustness of social systems, providing what is needed to achieve their goals under adverse conditions. However, diversity may create organisational capacity in excess of what is needed and reduce the system's efficiency, which possibly will conflict with business and political interests. This creates the efficiency vs. robustness dilemma. The development of Linux is presented as a case of a robust Complex Adaptive System software development, and compared with evolutionary and traditional engineering approaches. Linux was developed with no commercial interests, without any cost or schedule constraints and did not aim to be profitable. The efficiency vs. robustness of the development of Linux is then discussed within the context of what is expected of commercial software development. In conclusion, the paper argues that sacrificing efficiency for robustness can be a sensible approach for strategic and safety critical software intensive projects.

1. INTRODUCTION

Software intensive projects continue to experience schedule delays, cost overruns and reduced quality when the product is finally delivered. The causes of failure of large software intensive projects can be attributed to the complexity of today's technical systems and to the even more complex social systems that are required to engineer them. Planning and control of complex systems is a difficult task. The greater the complexity, the higher are the chances that the system will encounter situations that have not been considered.

Projects aim to be effective and efficient. Effectiveness is an indication of how well the project achieves its goals, and efficiency shows how much is wasted in the process. Complex systems need diversity to be able to adapt to unplanned conditions and achieving their goals. Diversity increases robustness of social systems, providing what is needed to achieve their goals under adverse conditions. Diversity may, however, create organisational capacity in excess of what is needed and reduce the system's efficiency, which possibly will conflict with business and political interests. This creates the efficiency vs. robustness dilemma.

The development of Linux is presented as a case of a robust complex adaptive system software development, and compared with evolutionary and traditional engineering approaches. Linux was developed with no commercial interests, without any cost or schedule constraints and did not aim to be profitable. The efficiency vs. robustness of the development of Linux is then discussed within the context of what is expected of commercial software development.

The history of failure of software intensive projects and the success of the development of Linux raise questions regarding the true nature of the development of complex software systems. In conclusion, the paper argues that models, parameters and engineering approaches currently in practice may not be adequate to deal with the complexity of today's technical systems, and that the social systems that engineer them are not robust enough to deal with the complexity of the task. Finally, the paper suggests that sacrificing efficiency for robustness can be a sensible approach for strategic and safety critical software intensive projects.

2. SOFTWARE INTENSIVE PROJECTS

Software intensive projects apply knowledge and engineering processes to develop products highly dependent on software. Software intensive products are in fact components of larger technical systems that are part of a solution to satisfy an expressed need. Often, software intensive projects engineer systems of strategic importance that impose strong schedule requirements, and safety critical systems that demand high quality. Defence systems, for example, are often strategic and safety critical.

Although many reasons have been offered to explain why software intensive projects fail, and many solutions have been proposed to address the problem, software intensive projects continue incurring schedule delays, cost overruns and delivering products with reduced functionality (DAF, 2000) (Standish, 1998-1, 1998-2).

Commercial software products, and even government strategic software developed by commercial organisations, are meant to be profitable. Commercial software projects impose constraints on schedule and

costs. Late delivery may not only cause disastrous strategic results, but also increases costs and reduces profit. The endless battle fought, and not always won, by software development managers is to keep the balance between costs, schedule and quality. Business pressures to cut costs may force managers to staff less experienced people. The result, however, is often the opposite: the quality of the software product decreases, the need for rework increases, which extends schedule and raises costs, confirming that the quality of software cannot be disassociated from the quality of people (Brooks, 1995).

Software developed under altruistic initiatives like “open source”¹ and “free software”² is not meant to be profitable, and unlikely to be constrained by schedule and budgets. These movements aim to develop software with higher quality, more flexibility and at lower cost, by harnessing the power of developers distributed all over the world. Organisations such as the Free Software Foundation (FSF) and Open Source Initiative (OSI) are motivated by ideals that aggregate followers that are willing to contribute with their expertise for free, for a cause and personal satisfaction. Without business constraints and with almost unlimited resources, software products developed as “open source” and “free software” are often useful and of good quality.

3. SOCIAL AND TECHNICAL SYSTEMS

Software intensive products are complex technical systems comprising of many interconnected parts, which require an as complex social system to engineer them. The social system comprises of people associated in teams and organisations, bounded by relationships, processes and contracts. The Law of Requisite Variety (Ashby, 1957) shows that to be effective, a system has to be at least as complex as the task it intends to execute (Bar-Yam, 2003), implying that technical systems have to be as complex as the need they should satisfy, and the social systems have to be as complex as the technical systems they are tasked to engineer.

Complex systems need diversity (or variety) to be able to adapt to unexpected conditions and continue achieving their goals. Diversity increases robustness of social and technical systems, allowing them to achieve their goals under adverse and unplanned conditions (Axelrod & Cohen, 2000) (Hitchins, 1992). Diversity in technical systems is present in the form of features that may be useful for unknown situations, or a spare computational capacity to expand the system's capabilities that might be required in the future. In social systems diversity comes in the form of human resources that provide knowledge, expertise and ideas.

3.1 Efficiency and Effectiveness

Effectiveness indicates how well a system achieves its goals. The effectiveness of a technical system is determined by its capacity of executing the task required to satisfy the need. The effectiveness of the social system represents its capacity of engineering the technical system.

Efficiency is an indication of how much waste the system produces to be effective. The efficiency of technical systems can be assessed by the ratio of energy produced and consumed and, for information systems,

the ratio between computational processing power and memory used and available. The efficiency of social systems that engineer technical systems can be evaluated based on the ratio of effort required to execute “end tasks” and “enabling tasks”, associated respectively to “end-products” and “enabling-products” (EIA, 1999).

To assess the effectiveness and efficiency of a real system, it should be compared with the ideal system (Hitchins, 1992) that is 100% effective and 100% efficient. The ideal technical system satisfies the need and does not waste energy nor computational resources. The ideal social system does not waste human and other resources. In the ideal social system every person knows what is needed to engineer the ideal solution, and no effort is spent on “enabling-tasks”, rework or learning.

Each system has a nominal efficiency to achieve maximum effectiveness (Hitchins, 2003, p.56). A system can be highly efficient and not being effective. Considering the number of problem fixed by unit of time, a software development team can be very efficient by addressing only the software problems that are easy to fix. If the critical and difficult problems are not resolved, however, the technical system may not be operational and the software development team would not be effective in achieving its ultimate goal.

Software intensive technical systems are not ideal. To account for unknown situations they sacrifice efficiency of computational resources to increase effectiveness of the system (e.g. the empirical rule of 30% of spare computational capacity is usually specified as a requirement). Similarly the social system that engineers the technical system is also less than ideal, and software intensive projects sacrifice efficiency to increase effectiveness. Lack of knowledge and human error are compensated by a number of planned “enabling-tasks” in the form of reviews and testing, and, not always planned, rework. Software development and engineering processes aim to increase effectiveness. Efficiency is determined by how well prepared are the people executing the process and engineering tasks.

3.2 The Efficiency Vs Robustness Dilemma

Software intensive projects aim to be effective and efficient. Being effective, the project meets the customer's expectations and delivers a product that satisfies the need. Being efficient, the project increases the value for money of the solution, reduces costs and increases profit.

As the complexity of the software intensive systems increases, both the technical and social systems need diversity to be robust to cope with situations that have not been anticipated. Robustness causes products to become more expensive due to features and resources that may be not immediately needed. The cost to engineer, develop and produce these sophisticated and robust products also increases to include enabling and redundant tasks to assure that the technical system will be effective and delivered on time.

Diversity of human resources is needed to increase the robustness of the social systems. A selection of domain experts of multiple engineering disciplines is needed to support the design and specification of software intensive systems. The development of software intensive systems often requires knowledge and expertise of hundreds of areas of knowledge, not always possessed by systems and software engineers. The more

¹ See Open Source Initiative (OSI) at <http://www.opensource.org/>

² See Free Software Foundation (FSF) at <http://www.fsf.org/>

experienced managers, engineers and technicians are, the more diverse and better prepared is the social system to address situations that have not been taken into account.

Diversity provides capacity in excess that may or may not be required. This spare capacity, however, may be the difference between success and failure. Increase in diversity is not directly seen as a benefit, and often perceived as a detriment that reduces efficiency, increases costs and lowers profit. Managers then face the efficiency vs. robustness dilemma to ascertain the risk of investing into diversity.

4. THE DEVELOPMENT OF LINUX

Linux is a Unix-like operating system developed as a collaborative task under the “open source”³ movement, which means its source code is freely distributed and available to the general public. The revolutionary method used to develop Linux produced a robust product that has proven to be of good quality, free of cost to the user, and became widely accepted to the point that it is threatening the interests of giants like Microsoft (Moody, 1998).

Linux is a complex technical system, developed by a not less complex social system: the Linux community. The success of Linux can be explained by insights from complexity theory, and may help to find strategies with better chances of success for software intensive projects.

4.1 The Development of Linux as a CAS

The development of Linux is an example of software development as a complex adaptive system (CAS), where software developers execute their tasks, interact and adapt without a central control. Three key processes provide the basis for adaptation: variety (or diversity), interaction and selection (Axelrod & Cohen, 2000).

Variety is present in the Linux community in the abundance of contributors, which possess two important attributes: motivation and knowledge. People that join the Linux community do it because they want and feel confident they can contribute with their knowledge, experience and skills. Their motivation may be driven by different reasons, whether making better software they can use, to expand their technical knowledge, or because they want to oppose to the hegemony of large business-driven corporations. Their goal, however, is the same and determined by the “open source” and “free software” initiatives.

The participants in the Linux community interact through a strict protocol and follow the etiquette of the “open source” community. Error reporting, suggestions and fixes are submitted via a well defined process. Selection occurs by the intervention of a select team that decides what will be included in the next release of Linux. The development process is highly redundant. Only what is judged to be good and adding value to the product is included, while probably a high amount of software code is rejected. Linux’s error reporting and selection processes are a form of Change and Configuration Management Control suggested by good software engineering processes. In case of Linux,

however, these processes may have emerged instead of imposed by planning in advance.

Linux is developed by volunteers, and there is no concept of individual gain, except for the satisfaction of doing what they want and believe, and perhaps to increase status in the community. The Linux community works as a team towards a goal that benefits the community and not just individuals, which is the essence of complex adaptive systems.

4.2 The Real Costs of Linux

The real costs of Linux and the effort spent in the development are unclear. The cost of Linux Kernel 2.6 is reported to be between \$50 million and \$176 million (US) and the cost of the whole Linux distribution, including Kernel, API Library and Core OS Utilities, is estimated in the order of \$1 billion (US) (Unknown Reader, 1998) (Wheeler, 2006).

The lower estimated cost for Linux Kernel 2.6 seems to be based on the average industry standard cost to produce the same number of SLOCs⁴ while the higher estimation assumes that if Linux were developed from scratch it would take at least the same effort as a similar commercial project estimated using the default COCOMO model (Wheeler, 2006).

Both estimations and their assumptions are unlikely to be valid for Linux. A horde of highly motivated programmers working part-time and for free is unlikely to have the same productivity of an average programmer in the industry. Also, the estimations seem not to take into account that the development of Linux is highly redundant and only a portion (the good one) of the code developed is selected to be included in the official release of Linux. The higher estimation, however, is an indication that the performance in the development of Linux could be at least 3.5 times less than the average industry standard. Taking into account the software code that is developed but not included in the product, the real cost of Linux development could be much higher, lowering even more the efficiency of the software development process of Linux, although there is no data to confirm this hypothesis.

4.3 Efficiency Vs Robustness of Linux Development

The “open source” software development approach worked well for the development of Linux, and created a complex adaptive system that is very successful and robust. The success can be seen by the quality of the Linux OS, its wide acceptance and the satisfaction of the members of the Linux community that continue offering their work for free. The social system is also robust. It adapts to changes that may occur in the pool of contributors and resists to threats that come from large organisations in the business world. For over 15 years the Linux community has survived and continue achieving its goal: to produce software of high quality, flexible and at low cost (at least for the users).

As the complexity of technical systems increases, the social system that engineers the solution needs to be robust to cope with the unknown, which in turn reduces the nominal efficiency of the system to achieve the intended effectiveness. The high robustness and

³ “Open source” does not refer to the Open Source Initiative (OSI). Linux is written and distributed under the GNU General Public License, under the banner of the Free Software Foundation (FSF).

⁴ 4,287.45 KSLOC (Weeler, 2006) equivalent to 33.5K FP (Jones, 1996, p.80), and the average industry standard cost of \$1.4K (US) for 1 FP (Jones, 1996, p.13, p.152).

possible low efficiency of the development of Linux confirm that systems that are robust trade robustness for efficiency (Hitchens, 1992, p.89).

5. STRATEGIES FOR COMPLEX PROJECTS

Strategic and safety critical systems are engineered by complex projects that need to meet quality, schedule and cost requirements. The search for strategies with better chances of success continues as software intensive projects still experience schedule delays, cost overruns and less than desired quality. The complexity theory offers insights that motivated approaches such as “evolutionary engineering” (Bar-Yam, 2003). The development of Linux, although highly inefficient, proved that “open source” is an effective and robust strategy for complex software products.

5.1 Strategies Supported by Complexity Theory

Evolutionary engineering adopts principles from complexity theory. It is based on incremental iterative changes, redundancy and parallelism, and fosters diversity. Traditional engineering, in contrast, constraints diversity by imposing standardisation, and adopts specific task assignments that reduces redundancy and parallelism (Bar-Yam, 2003). The characteristics of evolutionary engineering can be found in the development of Linux. Although likely to be effective, evolutionary engineering is also expected to be inefficient. Quality Function Deployment (QFD) (Akao, 1990) and Zachman Framework⁵ for enterprise architecture are tools that support evolutionary engineering principles and are indicated for the acquisition and development of complex technical systems for defence (Brown, 2000). These tools facilitate the solution to emerge through communication and cooperation between customers and providers.

“Open source” software development is a form of complex adaptive system that has shown to be effective for the development of Linux. Motivated and capable software developers in abundance, fueled by the absence of central control, provide the ingredients for diversity, redundancy and parallelism. An efficient communication process creates the collaborative environment that fosters cooperation and adaptation. Finally, an effective selection process guides the system to reach its goals and prevents from getting into chaos and eventually collapsing. All these ingredients are present in the successful complex adaptive system created by the Linux community.

The “open source” development approach, for obvious reasons, may not be indicated for the development of software components with security implications. “Open source” strategies, however, may improve the development of strategic and safety critical systems that often contain software components that are not under security classification. Many software components of defence systems (e.g. infrastructure, GUI, simulation and navigation systems) could benefit from the “open source” approach. Linux, for example, has been used as the operating system for military systems such as trainers and simulators⁶. By reducing the development

costs and improving effectiveness of software components that can be developed as “open source”, more resources could be applied to security and safety related components.

The low efficiency and free schedule characteristics of the “open source” approach oppose to the interests and constraints of commercial and strategic software development. However, the high effectiveness that comes from the complex adaptive system that develops the software product makes “open source” a strategy that can bring better results for the development of software intensive systems.

5.2 Trading Efficiency for Robustness

Complex systems need to be robust to cope with the unknown and adverse situations, and software intensive projects are not different. The more complex the technical and the social systems, the more diversity they must contain, which in turn increases exceeding capacity and lowers efficiency. Managers and decision-makers constantly face the efficiency vs. robustness dilemma.

The efficiency of software intensive projects should come from the quality of processes, people and organisations. The loss in efficiency to improve robustness can be compensated by investing in the quality of people and processes. Organisations that are better prepared to engineer complex software intensive systems should also be able to offer better commercial deals, without jeopardising their business interests.

Strategic and software critical systems ought to be effective and delivered on time. Technical systems of this nature often trade efficiency in favour of robustness and effectiveness. Evolutionary engineering supports this approach. To be able to engineer effective technical systems and deliver them on time, the social system also needs to be robust to be effective. Trading effectiveness for robustness is then an acceptable and better strategy for strategic and safety critical systems.

6. CONCLUSION

The success of the development of Linux raises questions regarding the true nature, costs and strategies used for complex software development. Are traditional engineering models used for strategic and safety critical software intensive systems adequate to deal with their complexity? Are the social systems that engineer them robust and effective? Is the efficiency expected of business enterprises compatible with the complexity of the task they have to execute?

As the complexity of problems that need software intensive systems increases, the complexity of the software intensive technical systems and of the social systems needed to engineer them also increases. Failure of software intensive projects can be associated with the complexity of technical and social systems and success thus depend on engineering models and strategies that better adapt to such complexity.

Complex systems are difficult to predict, plan and control, and need to be robust to cope with unplanned adverse situations. As the complexity of technical systems increases, the social system that engineers the solution needs to be diverse to cope with the unknown, which in turn reduces the nominal efficiency of the system to achieve the intended effectiveness. Diversity increases robustness and it is likely to reduce efficiency.

⁵ See <http://www.zifa.com/> - Framework Overview and Definition Architecture Quick Start.

⁶ Linux has been used as the operating system for single board computers and desktops for the Combat System Tactical Trainer (CSTT) ANZAC ship simulators, in used by the Royal Australian Navy (RAN) and the Royal New Zealand Navy (RNZN).

Strategic and safety critical software not only impose strong requirements on schedule and quality, but also on costs. Software intensive projects aim to be effective, to deliver products that meet the customer's expectations, and efficient, to increase value for money and profit. The diversity required to increase robustness reduces the efficiency of both technical and social systems, which is likely to conflict with political and business interests.

The "open source" software development of Linux is an example of a complex adaptive system, where the development of software is carried out without central control. Diversity, virtually unlimited resources, the absence of schedule and costs constraints, and an effective selection process, made Linux an effective and robust product. Not less effective and robust is also the social system that developed Linux. High effectiveness, however, is achieved by a system that is inefficient when compared with industry standards.

Each system has a nominal efficiency to achieve maximum effectiveness. The more complex is the system, the more diversity it must contain to be able to be robust when experiencing the unknown, which in turn lowers efficiency. Trading efficiency for robustness thus can be a strategy to assure the success complex projects such as required to develop strategic and safety critical software intensive systems. Organisations could, however, compensate the loss of efficiency when dealing with complex projects by investing in the quality of people and processes, which would also improve the organisation's ability to offer better commercial deals.

Better strategies that improve the chances of success of strategic and safety critical software intensive projects are needed, and strategies founded on concepts derived from complexity and systems theory seem to be promising.

REFERENCES

1. Akao, Y., editor, 1990, "Quality Function Deployment: Integrating Customer Requirements into Product Design", Productivity Press, Portland, OR.
2. Ashby, W. R. 1957, "An Introduction to Cybernetics", Chapman and Hall, London.
3. Axelrod, R., Cohen M. D. 2000, "Harnessing Complexity, Organizational Implications of a Scientific Frontier", Basic Books, New York, NY.
4. Bar-Yam, Y. 2003, "When Systems Engineering Fails – Towards Complex Systems Engineering", *International Conference on Systems, Man & Cybernetics, 2003, Vol. 2, 2021- 2028*, IEEE Press, Piscataway, NJ, 2003.
5. Brooks, F. P. 1995, "The Mythical Man-Month: Essays on Software Engineering", Anniversary Edition, Addison-Wesley Publishing Company, USA.
6. Brown, D. P., 2000, "Enterprise Architecture for DoD Acquisition", *Acquisition Review Quarterly*, Spring 2000.
7. Jones, C. 1996, "Applied Software Measurement", Second Edition, McGraw-Hill, USA.
8. DAF, Department of the Air Force, U.S.A., 2000, 'Guidelines for Successful Acquisition and Management of Software-Intensive Systems', Version 3.0, May 2000, viewed November 2003, <<http://web.nps.navy.mil/~menissen/mn3309/stsc-guidelines3.0/GSAMv3.pdf>>.
9. EIA, 1999, "EIA-632 Standard: Processes for Engineering a System", Electronic Industries Alliance.
10. Hitchins, D. 1992, "Putting Systems to Work", John Wiley & Sons, London, England.
11. Hitchins, D. 2003, "Advanced Systems: Thinking, Engineering and Management", Artech House, Norwood, MA.
12. Moody, G. 1998, "The Wild Bunch", New Scientist 160, no. 2164, December 12, 1998, p.42-46
13. Standish, G 1998, 'CHAOS: A receipt for success', Standish Group, viewed November 2003, <http://www.standishgroup.com/sample_research/PDFpages/chaos1998.pdf>.
14. Standish, G 2003, 'Latest Standish Group CHAOS Report shows project success rates have improved by 50%', Press Release, Standish Group, viewed 13 Nov. 2003, <http://www.standishgroup.com/press/>.
15. Unknown Reader, 1998, "The True Costs of Linux Development", The Register, 21st October 1998, viewed May 2007, http://www.theregister.co.uk/1998/10/21/the_true_costs_of_linux/.
16. Wheeler, D. A. 2006, "Linux Kernel 2.6: It's Worth More!", viewed May 2007 <http://www.dwheeler.com/essays/linux-kernel-cost.html>.

ABOUT THE AUTHORS

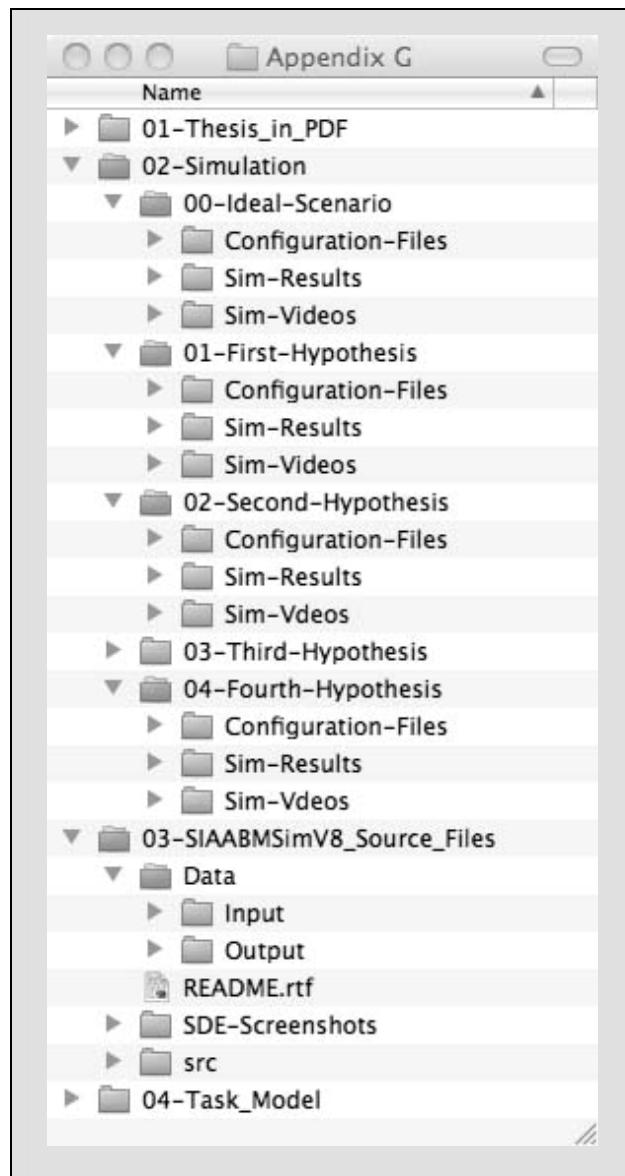
Ricardo Peculis - Mr Peculis is a senior systems engineer at Computer Sciences Corporation, where he has been involved with major defence projects as a senior engineer and technical manager. The challenge to succeed in software intensive projects motivated him to further investigate the causes that drive undesirable results and ways to improve the effectiveness of such projects. Mr Peculis is currently a PhD student at the Systems Engineering Evaluation Centre of the University of South Australia and his research addresses the implications of socio-organisational design into the complexity of software intensive acquisitions.

Derek Rogers - Dr Derek Rogers is an adjunct senior research fellow at the University of South Australia and Product Development Manager for BAE Systems. He focuses in the areas of systems engineering, wireless telecommunications, innovation and commercialisation. Dr Rogers has published approximately thirty papers and also holds an adjunct role at the University of Adelaide and visiting lecturer role at the University of Canterbury, New Zealand.

Peter Campbell - Professor A. Peter Campbell is currently (2004-7) working under contract with CSIRO Complex Systems Science Initiative to introduce complex system simulation tools for agricultural landscape planning and critical infrastructure analysis. Since May 2004, Professor of Systems Simulation and Modelling with the Systems Engineering and Evaluation Centre (SEEC) at the University of South Australia working on the application of complex adaptive system simulation technology to large scale system integration projects at UniSA and development of the Centre for Defence and Industry Systems Capability(CEDISC). Recent work includes development of agent based simulations to support organizational change within the Australian Defence Organization.

Appendix G: DVD Contents

The DVD included with this thesis contains this document in PDF, the source files of the SIAABMSim software⁴⁹, the configuration data used in the simulation and the simulation results. The DVD contains also a few illustrative screen movies of the SIAABMSim in operation. The file structure of the DVD contents is shown below.



⁴⁹ The source files of SIAABMSim are intended for reference only and can be used ‘as is’ for producing an executable of SIAABMSim for academic purposes. The README file contains instructions for compiling the software and the conditions for its use.

