

2011

Shall we play a game?

Craig Caulfield
Edith Cowan University

Follow this and additional works at: <https://ro.ecu.edu.au/theses>



Part of the [Higher Education and Teaching Commons](#)

Recommended Citation

Caulfield, C. (2011). *Shall we play a game?*. Edith Cowan University. Retrieved from <https://ro.ecu.edu.au/theses/447>

This Thesis is posted at Research Online.
<https://ro.ecu.edu.au/theses/447>

Edith Cowan University

Copyright Warning

You may print or download ONE copy of this document for the purpose of your own research or study.

The University does not authorize you to copy, communicate or otherwise make available electronically to any other person any copyright material contained on this site.

You are reminded of the following:

- Copyright owners are entitled to take legal action against persons who infringe their copyright.
- A reproduction of material that is protected by copyright may be a copyright infringement. Where the reproduction of such material is done without attribution of authorship, with false attribution of authorship or the authorship is treated in a derogatory manner, this may be a breach of the author's moral rights contained in Part IX of the Copyright Act 1968 (Cth).
- Courts have the power to impose a wide range of civil and criminal sanctions for infringement of copyright, infringement of moral rights and other offences under the Copyright Act 1968 (Cth). Higher penalties may apply, and higher damages may be awarded, for offences and infringements involving the conversion of material into digital or electronic form.

Shall We Play a Game?

A thesis submitted to Edith Cowan University
in fulfilment of the requirements for the degree
of Doctor of Philosophy (Computer Science)

By

Craig Caulfield

Student Number: 0985379

Supervisors: Dr S Paul Maj and Dr David Veal

Edith Cowan University
Faculty of Computing, Health and Science
School of Computer and Security Science

Date Submitted: 23 December 2011

USE OF THESIS

The Use of Thesis statement is not included in this version of the thesis.

Abstract

In response to real and perceived short-comings in the quality and productivity of software engineering practices and projects, professionally-endorsed graduate and post-graduate curriculum guides have been developed to meet evolving technical developments and industry demands. Each of these curriculum guidelines identifies better software engineering management skills and soft, peopleware skills as critical for all graduating students, but they provide little guidance on how to achieve this. One possible way is to use a serious game — a game designed to educate players about some of the dynamic complexities of the field in a safe and inexpensive environment. This thesis presents the results of a qualitative research project that used a simple game of a software project to see if and how games could contribute to better software project management education; and if they could, then what features and attributes made them most efficacious. That is, shall we— should we— play games in software engineering management?

The primary research tool for this project was a game called Simsoft. Physically, Simsoft comes in two pieces. There is an A0-sized printed game board around which the players gather to discuss the current state of their project and to consider their next move. The board shows the flow of the game while plastic counters are used to represent the staff of the project. Poker chips represent the team's budget, with which they can purchase more staff, and from which certain game events may draw or reimburse amounts depending on decisions made during the course of the game. There is also a simple Java-based dashboard, through which the players can see the current and historical state of the project in a series of reports and messages; and they can adjust the project's settings. The engine behind Simsoft is a system dynamics model which embodies the fundamental causal relationships of simple software development projects.

In Simsoft game sessions, teams of students, and practicing project managers and software engineers managed a hypothetical software development project with the aim of completing the project on time and within budget (with poker chips left over). Based on the starting scenario of the game, information provided during the game, and their own real-world experience, the players made decisions about how to proceed— whether to hire more staff or reduce the number, what hours should be worked, and so on. After each decision set had been entered, the game was run for another next time period, (a week, a month, or a quarter). The game was now in

a new state which the players had to interpret from the game board and decide how to proceed.

The findings showed that games *can* contribute to better software engineering management education and help bridge the pedagogical gaps in current curriculum guidelines. However, they can't do this by themselves and for best effect they should be used in conjunction with other pedagogical tools. The findings also showed that simple games and games in which the players are able to relate the game world to an external context are the most efficacious.

Declaration

I certify that this thesis does not, to the best of my knowledge and belief:

- Incorporate without acknowledgement any material previously submitted for a degree or diploma in any institution of higher education;
- Contain any material previously published or written by another person except where due reference is made in the text;
- Contain any defamatory material.

I also grant permission for the Library at Edith Cowan University to make duplicate copies of my thesis as required.

28-12-2011

Craig Caulfield

Acknowledgements

Special thanks are due to my supervisors Dr Paul Maj and Dr David Veal.

Paul was always generous with his time and knowledge, and was a source of wise counsel and encouragement over the long course of this project. His frank and reasoned criticism was sometimes like swallowing a bone; but it was also invariably right.

David could be relied upon to apply a meticulous eye to all aspects of this project— from making sure all university and ethical procedures were followed, to the experimental design, and to the papers that resulted. He was also instrumental in marshalling this thesis through its final, critical stages. All this was delivered with an unfailing sense of good humour

Without their guidance, this project would not have been completed.

Table of Contents

Use of Thesis	i
Abstract.....	ii
Declaration	iv
Acknowledgements	v
Table of Contents	vi
List of Figures.....	xi
List of Tables.....	xii
Chapter 1— Introduction.....	1
Background and Significance	1
The Software Crisis.....	1
The Complexity of Software Development.....	2
Addressing the Issues	3
One Possibility— Games	5
Defining Games	6
Problem Statement.....	10
Purpose Statement.....	11
Research Questions.....	12
Research Design Overview	15
Chapter 2— Literature Review.....	19
Introduction.....	19
The Nature of Software.....	20
The Essences of Software	20
The Accidents of Software	21
Preparing Software Engineers for a Complex Environment.....	25
The Profession of Software Engineering.....	25
The Education of Software Engineers.....	30
Software Engineering in a Social Environment	34

Dealing With Complexity and Change	36
Problem-Based Learning	39
Introducing Problem-Based Learning	39
Philosophical Under-Pinnings	40
Background and History	42
Criticisms of Problem-Based Learning	43
Is Problem-Based Learning Worth the Effort?.....	44
Games as an Implementation of Problem-Based Learning.....	45
History and Origins	45
How Games Are Used.....	50
The Rationale for Games.....	51
The Instructional Value of Games.....	53
A Systematic Survey of the Field of Games in Software Engineering Education.....	55
Summary	59
Chapter 3— Research Methodology	61
Introduction.....	61
Rationale for a Qualitative Research Design	61
The Researcher	62
The Research Sample.....	64
Ethical Considerations	65
Data Collection Methods	66
Simsoft Overview.....	68
Game Sessions.....	69
Game Administration	73
Pre- and Post-Game Surveys.....	74
Data Analysis and Interpretation	75
Reliability, Validity, and Applicability of the Findings.....	76
Reliability	77

Validity.....	77
Applicability.....	77
Limitations of the Study.....	78
Summary.....	79
Chapter 4— Findings	80
Introduction.....	80
Finding 1— There was evidence the participants were learning by doing.	81
Finding 2— Games such as Simsoft are not sufficient learning vehicles by themselves and need to be supplemented by other methods.....	83
Finding 3— Simsoft is a suitable pedagogical device for participants of different skills and backgrounds.....	84
Finding 4— The majority (49 out of 59) of participants said they would be prepared to invest greater time and effort in games such as Simsoft if the reward was deeper understanding of a problem domain.....	86
Finding 5— The majority (44 out of 59) of the participants found that working in groups was a positive experience.....	87
Finding 6— The majority (44 out of 59) of participants preferred playing a board game rather than a fully computerised game	88
Summary.....	89
Chapter 5– Analysis and Interpretation	92
Introduction.....	92
Analytic Category 1– Games and Learning.....	93
Learning in Simsoft.....	93
Learning Through Simsoft Compared to Others	94
Are Games More Effective Than Other Pedagogical Means?	95
Learning-Design Principles in Simsoft	97
Analytic Category 2– Games in Context	101
Context in Design.....	102
Context in Practice	105
No Game is an Island	106

Analytic Category 3– The Relative Complexity of Games	107
Related Work	109
Summary	113
Chapter 6– Conclusions and Recommendations	115
Introduction	115
Shall We Play a Game?.....	115
Long-Form Games as a Way of Creating Context.....	116
Games as Group Activities	117
Simple Games Can Be Effective.....	117
Recommendations.....	118
Recommendations for Educators and Trainers.....	118
Recommendations for Game Developers.....	118
Recommendations for Future Research	119
Final Reflections	120
References	122
Appendices	147
Appendix A: Causal Loop Diagrams	147
Appendix B: Stock and Flow Diagrams	150
Appendix C: Simsoft Game Board	153
Appendix D: Simsoft Instructions to Players.....	154
About Simsoft	154
Playing Simsoft	154
Statement of Work	155
Appendix E: Simsoft Instructions to Game Administrators	157
Appendix F: Information Letter to Participants.....	159
Appendix G: Informed Consent Document	161
Appendix H: Simsoft Database Design	163
Appendix I: Pre-Game Survey.....	164
Page 1. Introduction	164

Page 2. About You and Your Team	164
Page 3. About Project Management and Software Engineering	164
Page 4. Thank you	167
Appendix J: Post-Game Survey	168
Page 1. Simsoft and Problem-Based Learning Evaluation.....	168
Page 2. About You and Your Team	168
Page 3. About Games in General	169
Page 4. About Simsoft in Particular	170
Page 5. Test Your Knowledge.....	173
Page 6. Anything Else?	176
Page 7. Thank you	176
Appendix K: Simsoft Finding Review.....	177
Page 1. Simsoft Findings Review.....	177
Appendix L: Full Data Extract of Games Used in Software Engineering Education	180
Appendix M: Spatial Distribution of Games Used in Software Engineering Education	191
Appendix N: Review Studies of the Instructional Effectiveness of Games.....	192
Appendix O: Peer-Reviewed Conference and Journal Articles Stemming From This Research Project	195

List of Figures

Figure 1: Causal loop model of a typical project management development cycle. (The causal loop diagramming used in the model is explained in greater detail in Appendix A).	7
Figure 2: A system dynamics simulation of worker burnout. (The stock-and-flow diagramming used here is explained in more detail in Appendix B).....	8
Figure 3: The results of running the burnout simulation.	9
Figure 4: An idealised learning process incorporating games.....	12
Figure 5: Causal relationships amongst the components of recognised professions	27
Figure 6: Idealised learning process	39
Figure 7: Game surveys classified by game type, experiment type, and Bloom taxonomy	58
Figure 8: Units of work boxes on the left-hand side of the Simsoft game board. ..	70
Figure 9: Resource boxes on the right-hand side of the Simsoft game board	71
Figure 10: Simsoft dashboard.....	72
Figure 11: The process of analysing and interpreting the data.	75
Figure 12: Simsoft game board marked-up during a game session with an agile development term.	104
Figure 13: A section of one the game boards marked-up with players' notes and reminders.	108
Figure 14: A simple causal loop diagram	147
Figure 15: A stock and flow diagram of the classic predator-prey relationship...	150
Figure 16: Predator and prey oscillations.	151
Figure 17: Simsoft data model.....	163

List of Tables

Table 1: Standish Group CHAOS report benchmarks.....	2
Table 2: The principles of complex adaptive systems mapped against agile development practices (Meso & Jain, 2006, p. 23).	24
Table 3: Software engineering mapped against Ford & Gibbs (1996) components of professional practice.....	28
Table 4: SE2004 Project Planning and Project Personnel and Organization topics along with their Bloom (1956) classifications.....	32
Table 5: Problem-based learning compared to other active learning methods.....	40
Table 6: A comparison of quantitative and qualitative research designs.	63
Table 7: Comparison of players pre- and post-game test scores	82
Table 8: Participants responses when asked whether they thought Simsoft was easy or difficult to play	85
Table 9: Players' evaluation of game features	88
Table 10: Simsoft Compared with Gee's (2007a) Principles of Good Game Design	97

Chapter 1— Introduction

Background and Significance

The Software Crisis

In 1968 and 1969 NATO convened conferences of computer industry representatives and academics to help address what was seen as a growing gap between what was generally hoped for in complex software systems and what was actually achieved (Naur & Randell, 1969; Buxton & Randell, 1970). At the time it was recognised that the demands on software practitioners from industry, defence, and consumers would likely grow at an exponential rate. Yet, software engineering was then more of a craft than a profession (the term software engineering in the conference titles was considered deliberately provocative) and was already struggling to meet quality and performance measures; a software crisis in fact.

By 1982, it was estimated that 15% of all software projects failed to deliver anything, and cost over-runs of 100% to 200% were not uncommon (DeMarco, 1982, p. 3). In more recent times, the situation is still common:

For every six new large-scale software systems that are put into operation, two others are canceled. The average software development project overshoots its schedule by half; larger projects generally do worse. And some three quarters of all large systems are “operating failures” that either do not function as intended or are not used at all. (Gibbs, 1994, p. 86)

In the 1990s, despite some admirable successes such as the Sabre airline reservation system (Copeland et al., 1995) and the relatively uneventful passing of Y2K (Glass, 2000; Tipton, 2000; Yourdon, 2000; Crawford, 2001), software engineering quality and performance standards were still suspect (for example Baber, 1982, pp. 26 - 59; Sauer, 1993; Myers, 1994; Stix, 1994; Neumann, 1995; Applegate et al., 1996a; Applegate et al., 1996b; Barlas, 1996b, 1996a; Glass, 1998, 1999).

In more recent times, getting an accurate picture of the current state of the software crisis is difficult because companies are naturally reluctant to publicise failures and they may also oversell their successes. Recent Standish Group CHAOS reports into software project successes and failures (cited in Eveleens & Verhoef, 2010, p. 31) shows an improving trend over the last decade (Table 1), but these reports have

been criticised because the research methods and population they are based on are obscure (Glass, 2006; Emam & Koru, 2008; Eveleens & Verhoef, 2010).

Table 1: Standish Group CHAOS report benchmarks

Year	Successful (%)	Challenged (%)	Failed (%)
1994	16	53	31
1996	27	33	40
1998	26	46	28
2000	28	49	23
2004	29	53	18
2006	35	46	19
2009	32	44	24

In the absence of reliable data, it may be conceded that the net societal benefit of software has been positive, but the long and expensive history of software project and product failures continues to accrue new examples and influences how the profession of software engineering is perceived.

The Complexity of Software Development

If a software crisis really does exist, then it may be no surprise. From the early days of the industry, it was known that software is inherently complex, and this complexity is an essential rather than an accidental characteristic:

- Software needs to conform as best as possible to the arbitrary complexity imposed upon it by human institutions and systems (Brooks, 1995, p. 184). It is the usual case that these institutions and systems have been designed by different people with no underlying theme; still, software must be made to tie them together.
- Software “is pure thought-stuff, infinitely malleable” (Brooks, 1995, p. 185). This property is both seductive and dangerous: when change is needed it is likely that it will be easiest to change the software, but constant change, if not managed, can erode the integrity of the original design.
- Software is invisible and difficult to visualise: “we are hindered in our work by the fact that we cannot see our product and by the fact that we are neither guided nor constrained by the laws of physics, biology or chemistry in creating it and reasoning about it. Our product is a pure information product, being a

structure of information and relations upon that information” (Osterweil, 1987, p. 3).

Then, how we come together to build this complex object adds yet another layer of complexity. It has been shown that:

... the decisions that people make in organizations and the actions they choose to take are significantly influenced by the pressures, perceptions, and incentives produced by the organization’s planning and control system(s)... In particular, knowledge of project schedules was found to affect the real progress rate that is achieved, as well as the progress and problems that are reported upward in the organization. (Abdel-Hamid & Madnick, 1983, p. 341)

So, in a complex societal system, such as a software development project, building a complex product, unappreciated causal relationships, dynamic complexity, and structural delays may lead to counter-intuitive outcomes of seemingly sensible decisions (Forrester, 1975; Abdel-Hamid & Madnick, 1983, p. 341). Without thoughtful planning and execution, poor quality software may be the result.

Addressing the Issues

There are some key indicators that the field of software engineering is trying to address these issues. A software engineering body of knowledge (SWEBOK) has been defined to characterise the contents of software engineering and to provide a foundation for curriculum development (Bourque et al., 1999); there are now professional accreditation and certification programs by which members of the field can be assessed (Naveda & Seidman, 2005); and professionally-endorsed curriculum recommendations have been developed to meet technical developments and evolving industry demands. Of these latter, the following are representative:

- Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering (SE2004) (Joint Task Force on Computing Curriculum, 2004).
- Curriculum Guidelines for Graduate Degree Programs in Software Engineering (GSWE2009) (iSSEc Project, 2009).
- Curriculum Guidelines for Undergraduate Degree Programs in Information Systems (IS2010) (Joint IS2010 Curriculum Task Force, 2010).

Each of these curriculum guidelines mentioned above identifies better software project management skills as critical for all graduating students, but they provide

little guidance on how to achieve this. Recognising that competent software engineering students need to supplement the abstract, theoretical side of their studies with some form of practical experience, educational institutions have typically used practicums where the students work in small groups to take a product idea from conception, through design, building and testing, to final delivery. These practicums can be delivered in a number of ways:

- Capstone projects: these are projects designed to synthesise what the students have learned so far and give them a practical way to exercise their skills. The projects themselves may be instructor-designed or proposed by industry and usually cover the final semester of the course (Brereton et al., 2000; Cheng & Lin, 2010).
- Work placements and sandwich courses: students are placed with software companies where they participate in real projects as paid employees. These placements may happen in the later parts of the student's course and may be single opportunities, or intertwined— sandwiched— over a longer period (Lay et al., 2008; Ribaud & Saliou, 2008).
- Laboratories: student teams work for extended periods on large-scale, ongoing projects within a standardized and evolving development process, which can accommodate team members leaving and joining (Sebern, 2002).

Often, these practicums come near the end of the students' studies, where they can tie together any loose threads by allowing the students to practice what they have learned. "However, this appears to be too little, too late. Projects are often only a single semester in length, students do not benefit from the integration of ideas and practice until the end of their studies, and team orientation is often undermined by scholastic competition for grades" (Schlimmer et al., 1994, p. 213).

While the practicums are designed to give students an opportunity to apply their knowledge in a practical way, they often fail because the students are overloaded with many conflicting concerns and often "aren't mature enough to appreciate the importance of many SE topics. On one hand... pay attention to documentation, apply configuration control, test thoroughly... On the other hand, our students have difficulty appreciating issues— such as team organization and cost estimation— that software professionals know from the trenches" (van Vliet, 2006, p. 56).

One Possibility— Games

In the previous section it was shown that the various software engineering and information systems curriculums place great emphasis on making sure graduates are cognisant of the value of sound software project management, but they provide little guidance on how to practically achieve this. Given that software development projects are complex socio-technical systems then arguably what is needed is an instructional method that provides students with an opportunity to experience the dynamics of a software project in something akin to a real-world environment— as Confucius said, “I hear and I forget, I see and I remember, I do and I understand”.

But, experience— Confucius’ doing— can be expensive. There is a story of a young IBM executive whose innocent mistake caused a \$10 million loss for the company. Coming before Thomas J Watson, the formidable IBM boss, the contrite executive said, “I’m here to tender my resignation”. Watson replied, “You must be kidding! We’ve just spent ten million dollars training you” (Awad & Ghaziri, 2008, p. 281).

The young IBM executive was lucky to have an enlightened boss, but must things always happen this way? Must mistakes be made in the real before we can learn from them? Perhaps not: games are a way of ‘doing’ in a controlled and inexpensive way so that software engineers and software project managers don’t repeat the same expensive mistakes (cost and time over-runs, dissatisfied end-users, burnt out staff, unstable or unreliable software) that bedevil modern software projects.

Of course, games aren’t the only way of achieving this, but:

- Games have been used as learning tools in many different business, military, and social environments, and have proven to be efficacious (Perla, 1990; Schrage & Peters, 1999; Michael & Chen, 2005; Gee, 2007a; Prensky, 2007).
- Games draw their intellectual integrity from a number of sources including educational theory (Dewey, 1938/1963; Lewin, 1952; Papert, 1980; Kolb, 1984), operations research (Thomas & Deemer, 1957; Wilson, 1968, pp. 36 - 50), small-group behaviour research (Kennedy, 1971b, 1971a), war-gaming, decision sciences (Mayer, 2009, p. 827), and systems engineering (Raser, 1969, pp. 46 - 55), and problem-based learning (Savin-Baden & Major, 2004).

So, games have a pedigree to be taken seriously as research and pedagogical tools. First, games need to be more formally defined.

Defining Games

To play a game, “is to engage in activity directed towards bringing about a specific state of affairs, using only means permitted by specific rules, where the means permitted by the rules are more limited in scope than they would be in the absence of the rules and where the sole reason for accepting such limitation is to make possible such activity” (Suits, 1967, p. 156). This classic definition contains the key elements common to all games:

- A goal or objective: a “specific achievable state of affairs” (Suits, 2005, p. 186), such as crossing the line first, scoring the most points, or having the best hand. Goals or objectives differentiate games from other types of play. For example, if a game doesn’t have a goal “but is something that can be just played with in many ways depending on your whim, you have what they refer to as a toy” (Prensky, 2007, p. 120).
- Means: the legal or legitimate ways of trying to achieve the goal or objective of a game. Using a weapon in a boxing match is one way of achieving the goal of downing an opponent, but it is, of course, illegal (Suits, 2005, p. 187).
- Rules: the legitimate means of achieving the goal of a game. Often, rules gratuitously prohibit the most efficient means of reaching a goal in order to make a game challenging and engaging (Suits, 2005, p. 187): a golf ball *could* simply be placed in a cup; instead, it *must* be hit from a distance and played from where it lies along the way.
- Lusory attitude: a free-willed acceptance by players of the conceit created by seemingly arbitrary rules simply in order to participate in the game (Costikyan, 2005, p. 195; Suits, 2005, pp. 188–189; Prensky, 2007, pp. 123 – 124).

In summary, Suits offers a portable version of a game: “the voluntary attempt to overcome unnecessary obstacles” (2005, p. 190).

With this definition in mind, a game can be said to be different from a model or simulation. To start, a model is:

in the long run.

A model is almost guaranteed to be incomplete because it is an abstraction of reality; the exact components of reality included in the model will depend on what the user is trying to explore. Even so, a model “saves us from a certain self-deception. Forced into the open, our ideas may flutter helplessly; but at least we can see what bloodless creatures they are. As inquiry proceeds, theories must be brought out into the open sooner or later; the model simply makes it sooner” (Kaplan, 1973, pp. 268 - 269).

Meanwhile, a simulation is a special kind of model that exhibits processes in some way like the system it represents, and that shows how these processes change from state A to state B, between two points in time (Miller, 1978, p. 83). Consider, for example, the simulation in Figure 2 which represents worker burnout as described by Homer (1985).

Burnout begins when a person working on a project tries to meet unmet expectations by working longer hours. By working longer hours they are exposed to more of the normal stress of work and consequently their finite store of “adaptive energy” (Selye, 1974, 1978) is depleted more quickly and they also have less time to recover. This depleted energy level may leave the person even less capable of meeting their expectations, or may cause them to make mistakes that have to be fixed at the expense of real progress. In response, they may try to work harder, which will deplete their energy levels still more. Unless the person is granted some respite, this vicious cycle may continue until they leave in frustration or they are burned out and no longer able to contribute to the project.

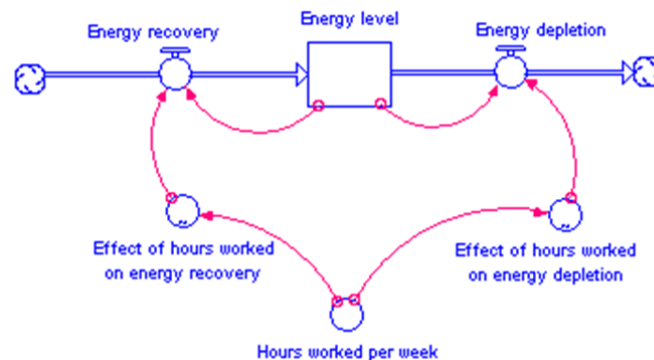


Figure 2: A system dynamics simulation of worker burnout. (The stock-and-flow diagramming used here is explained in more detail in Appendix B).

When this burnout simulation is run, a multi-scale graph is produced (Figure 3 **Error! Reference source not found.**).

Viewed over a 13-month period, the person starts out by working a 40-hour week. Every couple of weeks there is a spike and they have to work 50-hour weeks for a short time (this pattern can, of course, be changed to model any real-world circumstance). The graph shows that the person's energy levels rise and fall in line with oscillations in the work week, but the overall trend is downwards because the constant spikes in work never allow enough time for proper recovery.

Building a simulation such as this continues the explication begun when a theory is turned into a model:

Turning a model into a simulation, of course, brings this process one step further, for functional relations must additionally be specified and defined. Simulation-construction thus functions, as does any theory-construction, to systematize and order empirical findings, but in addition it *disciplines theory*, since concepts must be explicitly defined, and more importantly, relations among the elements must be completely specified if the simulation is to “run” or cycle. (Crow cited in Raser, 1969, pp. 73 - 74)

Simulations are different from games, but the distinction is subtle:

Both are mathematical models, but they differ in purpose and mode of use. Simulation models are designed to simulate a system and to generate a series of statistical results regarding system operations. Games are also a

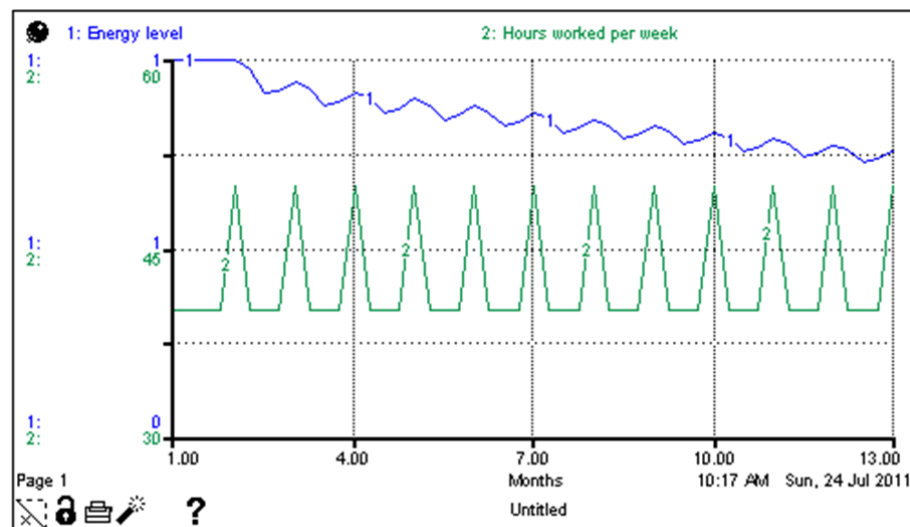


Figure 3: The results of running the burnout simulation.

form of simulation, except that in games human beings play a significant part. In games, human beings make decisions at various stages and games are distinguished by a sense of play. Major goals of game play are to improve decision-making skills and to facilitate an understanding of the game environment simulated by participation of the players (Shim, 1978, p. 26).

Therefore, as used here, a model is a convenient representation (in words, numbers, or other symbols) of some real-world, socio-economic or socio-technical system; a simulation is a dynamic, operational model through which changes over time are revealed; and a game is a simulation that is purposefully run, wholly or partly determined by players' decisions, within some predetermined context. In a game, there is something at stake (Huizinga, 1971, p. 49).

Games naturally come in many forms. In a seminal work in the field, *Man, Play and Games*, Caillois (1961) proposed a classification that depends on whether the role of competition (*agôn*), chance (*alea*), simulation (*mimicry*), or vertigo (*ilinx*) is dominant. *Agôn* are those games “that would seem to be competitive... like a combat in which equality of chances is artificially created in order that the adversaries should confront each other under ideal conditions” (Caillois, 1961, p. 14). Football, billiards, or chess fall into this category. *Alea* are games of chance such as roulette or a lottery; games of *mimicry* involve the players becoming other characters, such as cowboys and Indians; while *ilinx* are “those which are based in the pursuit of vertigo and which consists of an attempt to momentarily destroy the stability of perception and inflict a kind of voluptuous panic upon an otherwise lucid mind” (Caillois, 1961, p. 23).

The games that this research project deals with are a subset of Caillois's *agôn* classification and they use an adjective— serious— to show they want for more than simple amusement and that they are designed to educate, train, or inform their players (Abt, 1970; Schrage & Peters, 1999; Michael & Chen, 2005).

Problem Statement

Not all is well in software engineering.

Despite many admirable successes, some of the brightest minds, and many decades of experience, many software products are buggy, more expensive than they need to be, fragile, and sometimes life-threatening. Hard evidence to call this a software

crisis is difficult to gather, but it may be conceded that the demands placed on software engineers will continue to grow and that room exists to improve both the quality of software and other key project management processes.

Nevertheless, the profession is maturing and trying to do things better by:

- Defining a body of knowledge, the SWEBOK, to characterise the field and form the basis of curriculum development and accreditation, licensing and certification programs.
- Creating standards of ethics and conduct to guide software engineers in responsible behaviour.
- Developing professionally endorsed curriculum guidelines for initial and ongoing education to make sure that software engineers have the right skills and knowledge to deal with a complex and dynamic domain.

The focus of this research project is on this last item because it influences so many other areas of software engineering. Curriculum guidelines say time and effort have to be devoted to software engineering management, but, apart from traditional lectures, workshops, projects, and capstone projects, they don't say how this should be taught. Perhaps it is because in software engineering management we see the full face of a complex, dynamic, and human-centred socio-technical system.

The solution posited here is that serious games can help. For this research project, a game called Simsoft was developed (Caulfield et al., 2011a; Caulfield et al., 2011c; Caulfield et al., 2011d) to see what contribution its design features could make to the education of software engineers and software project managers and thereby fill some of the pedagogical gaps in the SE2004, IS2010, MSIS2006, and GSwE2009 curriculum guidelines.

Purpose Statement

The purpose of this research project is to see if and how games can contribute to better software engineering management education by helping software engineers and project managers explore some of the dynamic complexities of the field in a safe and inexpensive environment. If games *can* contribute, then what features make them most efficacious?

Research Questions

Here is Edward Bear, coming downstairs now, bump, bump, bump, on the back of his head, behind Christopher Robin. It is, as far as he knows, the only way of coming downstairs, but sometimes he feels that there really is another way, if only he could stop bumping for a moment and think of it. And then he feels that perhaps there isn't. (Milne, 1998, p. 1)

So it must also be for many software project managers dealing with tight deadlines, budget concerns, quality issues, production crises, and unexplored possibilities: so many things to consider but little pause to do so very meaningfully.

Managers in such situations often resort to silver bullets, those tools that promise an order-of-magnitude improvement in programmer productivity. The list of such tools is long and over the years has included different development methodologies, object-oriented techniques, CASE tools, prototyping, software reuse, process improvement, and specific languages (Cox, 1990; Davis, 1993). Since none of these tools has delivered on the promise of vastly improved productivity, Brooks (1995, p. 219) would call each just a brass bullet.

Neither silver nor brass, there may be another way to address such issues. In many different fields, games have developed into a mature instructional technique based on the idealised learning process shown in Figure 4 (Sterman, 2000, p. 34).

Within this feedback structure, we receive information in its many forms from the

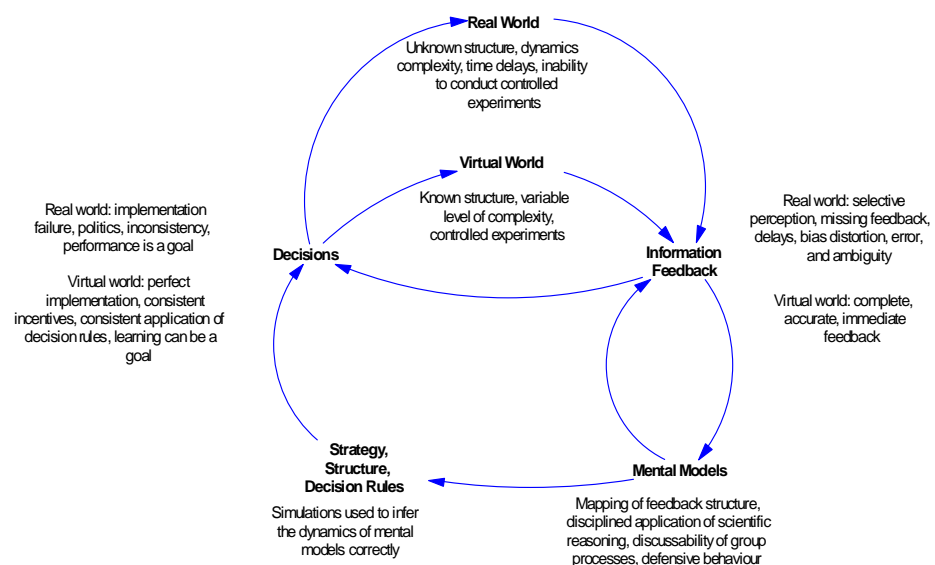


Figure 4: An idealised learning process incorporating games.

real world in which we live, yet this information can be incomplete, biased, delayed, or in other ways distorted. Still, based on this information, we make decisions that are in turn filtered through our existing mental models, in the process changing or confirming the structure of our real-world systems and creating new decision rules and new strategies or reinforcing the existing. The process then repeats against this new baseline. Virtual worlds, or games, act as an alternative to applying our decisions to the real-world, a way of quickly, inexpensively, and consistently experimenting with different ideas and thereby increasing our store of contexts.

Whether games can indeed do this in software engineering management gives rise to the first research question:

Can games contribute to better software engineering management education? (Q1)

Naturally, games are not the only way in which a contribution could be made to better software engineering education. Besides traditional lectures, laboratories, case studies, a number of organisations and academic institutions in recent years have initiated training programs that aim to more fully round the theoretical background of software engineers, often mixing this with real-world projects (Dawson et al., 1992; Mathiassen et al., 1999). Yet, for the most part “students are learning their real world awareness in industry, working on real projects where their mistakes affect all around them” (Dawson et al., 1997, p. 287). How games relate to more traditional pedagogical tools gives rise to the hypothesis associated with research question Q1:

Games built on sound software project management principles are a more effective means of improving software project management education than more traditional pedagogical means. (H1)

In general, games are more administratively time-consuming and complex to use in education. So, if they *can* contribute to better software engineering management, then we need to determine what features and attributes make them most efficacious in order to get the most from them. This gives rise to the second research question:

If games are to contribute to better software project management education, what features make them most efficacious? (Q2)

Software is often developed within an ill-structured environmental context (Bostrom & Heinen, 1977a, 1977b; Keen, 1981; Kling & Iacono, 1984; Hirschheim & Klein, 1989; Bennetts et al., 1998; Day, 2000) that includes sometimes contradictory human and business priorities, meaning that a purely technical or logical solution is not always the best guarantee for success. A necessary step is therefore to determine what factors help or hinder in this regard, giving rise to the first hypothesis in support of research question Q2:

For best effect, players should be able to easily relate the context of a game to their real-world experience. (H2.1)

From the literature of the field it can be said that the software development process has been *modelled* (Belady & Lehman, 1976; McCabe, 1976; Remus & Zilles, 1979; Boehm, 1981) and *simulated* (Abdel-Hamid & Madnick, 1991; Variale et al., 1994; Hansen, 1996; Madachy, 1996; Tvedt, 1996; Collofello, 2000; Martin & Raffo, 2001) many times. These existing models and simulations have captured many of the essences of software engineering practice and would naturally serve as a starting point for game development, yet many have not presented the field's body of knowledge in an intuitive way that encourages learning. For example, perhaps the most well-known simulation (Abdel-Hamid & Madnick, 1991) contains over 300 underlying variables, doesn't have a way to interact with the model except through direct manipulation of these variables, and yet does not describe the development process in detail (Martin, 2002b, pp. 32 - 37).

Set against these models is research that has compared the learning outcomes between a range of simple and more complex games. While the most complex game offered "the richest learning experience available, the game's very formidable appearance probably intimidated a number of players or forced them into a learning situation they were unprepared or unwilling to negotiate" (Wolfe, 1978, p. 152). The next most effective game in Wolfe's study was found to be the least complex, supporting similar research that showed relatively simple games can provide essentially the same benefits as the more complex (Raia, 1966, p. 351; Watt, 1977, p. 2; Butler et al., 1979). Therefore, making games only as complex as absolutely necessary, or hiding unnecessary detail, could be a way of achieving the

best learning outcomes while avoiding the player mortality (boredom and dropout) noted by Wolfe. This gives rise to the second hypothesis in support of research question Q2:

For best effect, games should be simple to play and understand and only as theoretically complex as needed to explore the concepts at hand. (H2.2)

* * *

[Christopher Robin] picked his bear up by the leg and walked off to the door, trailing Winnie-the-Pooh behind him... and in a moment I heard Winnie-the-Pooh— bump, bump, bump— going up the stairs behind him (Milne, 1998, p. 145).

It seems that Pooh Bear never did get pause to consider a better way of tackling those stairs. Perhaps the same metaphorical stairs won't prove to be so painful and inscrutable for software engineers.

Research Design Overview

The primary research tool for this project is a simple game called Simsoft. Teams of players are given a simulated project to operate from start-up to the delivery of its objectives. Based on the starting scenario of the game, information provided during the game, and their own real-world experience, the players make decisions about how to proceed— whether to hire more staff or reduce the number, what hours should be worked, and so on. After each decision set has been entered by the players, project time is advanced by a week. The game is now in a new state, which the players must interpret from the state of the board and decide what to do next.

Physically, Simsoft comes in two pieces:

- An A0-sized printed game board (see Appendix C: Simsoft Game Board) around which the players gather to discuss the current state of the project and to consider their next move. The board shows the flow of the game while plastic counters are used to represent the staff of the simulated project. Poker chips represent the budget the team, with which they can purchase more staff, and from which certain game events may draw or reimburse amounts depending on decisions made during the course of the game.

-
- A simple Java-based dashboard (see for example, Sterman, 1988; Langley et al., 1999; Caulfield et al., 2011b), through which the players can:
 - See the current and historical state of the project through a series of simple reports, messages, and other information.
 - View the underlying system dynamics model so they can be fully informed about the relationships behind particular game variables (Machuca, 2000).
 - Can adjust the project's settings, for example to recruit new staff, before advancing the game's time to create the state of the project.

The aim of the game is to complete the project on time and with funds (poker chips) left over.

The engine behind Simsoft is a system dynamics model which embodies the fundamental causal relationships in software development projects. System dynamics has been used for this task before and a case has been presented in other work (Caulfield, 2001; Caulfield & Maj, 2001, 2002, 2007). In line with similar efforts (Abdel-Hamid & Madnick, 1991), this engine will draw its rules from the software engineering body of knowledge (Bourque et al., 1999).

System dynamics is a modelling approach to dynamic socio-technical problems, stemming from the work of Forrester (1961, 1969, 1971) at MIT and since developed (Wolstenholme, 1990; Sterman, 2000; Senge, 2006), that allows a modeller to mix soft variables (morale, perceptions, motivations) with familiar hard variables (time, cost, resources). A system dynamics model is not so much a tool for time-point prediction, but more of an experimental device to see how certain variables might change over time under the influence of unappreciated causal relationships, dynamic complexity, and structural delays. The end result is hopefully a more informed mind set with which to manage the situation at hand (Caulfield & Maj, 2002).

Behind the system dynamics model will be a relational database (see Appendix H: Simsoft Database Design) to store the decisions entered by the players, the parameters which define the particular project (for example, budget and time), and which will capture the state of the model at each time slice. This will allow the game to be rolled backward or forwards, replayed, and studied.

Games such as Simsoft have typically been deployed in university courses in one of three ways:

- Teach sections of the same course using different methods, and then compare the results of students on a common test. For example, all students in a course might attend a common lecture, and then attend either a traditional tutorial session or a tutorial that uses a game (see, for example McKenney, 1962; Raia, 1966). Besides the difficulty of obtaining adequate control of factors such as student composition, instructor characteristics, and grader evaluations, previous studies in this vein have shown that students participating in games are obviously learning ‘something’ other than what the traditional method might teach them, but that ‘something’ cannot be measured by a common test (Parasuraman, 1981, p. 192).
- Evaluate the student’s grades or scores in the simulation part of the course with their grades in other more traditional assignments and examinations in the course (see, for example Remus, 1977; Remus & Jenner, 1981). There are some conceptual and methodological problems with this approach. For example, a student’s game score might reflect their ability to play or beat the game rather than their decision-making ability (Parasuraman, 1981, p. 194).
- Sample players subjective attitudes regarding the usefulness of the games before, during, or after play, or a combination thereof, by obtaining written feedback (see, for example Jackson, 1959; Dill & Doppelt, 1963; McKenney & Dill, 1966; McKenna, 1991; Herz & Merz, 1998). This technique can be criticised on the basis of “how qualified are college students, with little or no practical business experience, to make any meaningful evaluation of business simulation games?” (Parasuraman, 1981, p. 194). Other studies have shown that student performance in games, when compared to that of experienced managers, raises serious questions about how much can be generalised to behaviour patterns in the business world (Babb et al., 1966, p. 471).

Each approach therefore has its strengths and weaknesses and is part of a broader debate concerning the value of games as pedagogical devices in themselves and when compared to other methods of instruction (Amstutz, 1963; Moore, 1967; Boocock, 1970; Moskowitz, 1973; Hand & Sims, 1975; Wolfe & Guth, 1975; Parasuraman, 1981; Remus & Jenner, 1981; Prohaska & Frank, 1990).

To address some of these issues, an approach similar to the third option above will be followed. Simsoft game sessions will be conducted for teams of post-graduate project management students (for software and general projects), and practising software project managers and developers. Overall game evaluation will be made on the basis of pre- and post-games surveys, performance in Simsoft, and a qualitative rich analysis of the interactions that were observed during the game sessions (Drappa & Ludewig, 2000).

Chapter 2— Literature Review

Introduction

The purpose of this research project is to see if and how games can contribute to better software engineering management education by helping software engineers and project managers explore some of the dynamic complexities of the field in a safe and inexpensive environment. If games *can* contribute, then what features make them most efficacious? To do this, a critical review of the current literature was necessary and this continued throughout the data collection, analysis, and synthesis stages of this project.

During the course of this review, three main topic areas became apparent:

- The inherently complex nature of both software and the way it is developed.
- The way software engineers are educated to work in this type of environment and what lessons we might be able to learn from the way others have sought to handle similar situations.
- Problem-based learning, in general, and games, in particular, as an alternative way of making software development more tractable.

Looking at the inherent complexity of software and the development process used to produce it sets the context for how well (or not) we now educate software engineers. The literature of problem-based learning is reviewed for the insights it provides into how we might teach those who will be working in complex, dynamic, and ill-defined environments such as software development. Within this, the literature of games is reviewed to see how this practical implementation of problem-based learning theory is relevant to improving software engineering education.

To conduct this literature review, many different sources were used, including books, professional journals, periodicals, dissertations, and Internet sources, plus many extant games. Throughout the review, important gaps and omissions in the relevant sections of the literature are suggested, and any contested areas and issues are explored. The summary at the end of this chapter shows how the review of the literature has contributed to the design and conduct of the research.

The Nature of Software

In “No Silver Bullet” Brooks (1987) discusses the Aristotelian *essences* and *accidents* of software engineering while looking for the same order-of-magnitude improvement in productivity seen in hardware development— a failed search, ultimately. The former are the inherent complexities of software (the abstract data types, relationships among data items, algorithms and invocation of functions), while the latter are the difficulties that attend its production (representing abstract entities in programming languages, or building the software in a team environment). To date, most of the significant gains in software productivity have come by making accidental tasks less burdensome and error-prone, for example through high-level languages and sophisticated development environments. Brooks believed that more research into the essences of software had the potential to yield steady and modest productivity improvements— the best that could be hoped for since there was no realistic silver bullet.

Apart from pointing to areas of fruitful productivity research, the essences and accidents dichotomy also provides a useful means with which to look at the nature of software.

The Essences of Software

Some of the essences of software have already been mentioned in the Introduction. For example, software artefacts are naturally complex because they are intangible and difficult to visualise, and this complexity is the root cause of many other problems:

From the complexity comes the difficulty of communication among team members, which leads to product flaws, cost overruns, schedule delays. From the complexity comes the difficulty of enumerating, much less understanding, all the possible states of the program, and from that comes the unreliability. From the complexity of the functions comes the difficulty of invoking those functions, which makes programs hard to use. From complexity of structure comes the difficulty of extending programs to new functions without creating side effects. From complexity of structure comes the unvisualized states that constitute security trapdoors. (Brooks, 1987, p. 11).

Adding to the complexity, software has no fundamental theory (Osterweil, 1987, p. 3), like the law of physics, with which we can reason about its behaviour:

Because computer programs are in essence mathematical objects whose values are constructed from bits, software programs are discrete

(particlelike) rather than continuous. A mechanical engineer can stress a component with a large force and assume that if it survives it will not fail when subjected to a slightly smaller force. When an object is subject to the (mostly continuous) principles of the physical world, a small change in one quantity generally produces a small change in another. Unfortunately, no such generalities apply to software: one cannot extrapolate between test cases. If one chunk of software works, that fact says nothing about the operations of a similar chunk of code; they are discrete and separate. (Jackson, 2006, p. 62)

This makes it difficult to thoroughly test software without actually building it and running it in a live environment (Kruchten, 2005) with all the attendant risks this involves.

Software must also conform to the arbitrary design of the human institutions and processes in which it is deployed and accept change because in a system of software, hardware, and humans, it is the most malleable (Brooks, 1987, p. 12). These are naturally properties that organisations want to take advantage of, but constant change, if not managed, can erode the integrity of the original design, and when combined with relatively low manufacturing costs, can lead to shortcuts:

Program implementation is more like preparing a cast in mechanical engineering... The real “manufacturing” of software entails almost no cost; a CD-ROM, for example, costs less than a dollar, and delivery over the Internet only a few cents. Often it doesn’t matter if the design... is a bit wrong; we can just fix it and manufacture it again... You can’t do that with a bridge or a car engine because the cost would be huge, and that forces engineers involved in building these things to get them right the first time. (Kruchten, 2004)

Because software is complex, difficult to reason about and test, and yet cheap and easy to change, it is perhaps understandable that many implementations are not right the first time, if at all.

The Accidents of Software

To recap, an accidental feature of software is something that attends its production such as the programming language it is written in or the development process by which it is created.

Taking the latter as an example, the type of development process by which software is created or maintained typically falls somewhere on a continuum

between heavyweight and lightweight. Heavyweight processes emphasise detailed upfront planning, well-defined roles for each team member, and specific linear development phases— all of which is carefully documented and reviewed. For example, the waterfall process consists of two basic phases— analysis and coding— surrounded several overhead phases— system and software requirements definition, program design, and testing— which are designed to bring discipline to an otherwise complex undertaking. Iteration is possible, but usually only between the immediately preceding and succeeding phases. In this waterfall, program design is critical and it must be documented in detail so it can be shared by the whole team (Royce, 1970; McConnell, 1996, pp. 136 – 139; Royce, 1998, pp. 6 – 11).

Meanwhile, lightweight, or agile, development processes emphasise iterative and incremental development cycles and the delivery of working software over strict adherence to processes, plans, and documentation (Larman & Basili, 2003; Cockburn, 2006; Larman, 2006). Under Scrum (Schwaber, 2004), for example, there are also specific project roles, but these are less prescriptive than under the waterfall process. There must be a client representative co-located with the development team and this person must have the knowledge and authority to resolve most issues quickly when they arise. Managing the process is a Scrum Master, whose job is to remove any impediments and to buffer the team against any distractions. Then, from a product backlog, or list of work items, the team (which includes the customer representative) decide what can be achieved in the next sprint— a time-boxed development cycle of between 2 and 6 weeks. At a short daily stand-up meeting, the team members tell each other what they achieved yesterday, what they plan to do today, and report any road blocks they may have encountered. Because progress is measured by working software rather than by comparison to a project plan, the team need to demonstrate what they have achieved at the end of each sprint.

Naturally, heavyweight and lightweight processes have advantages and disadvantages and some are better suited to particular situations than others.

For example, the traditional waterfall development cycle of requirements definition, analysis, design, code, and testing needs the requirements to be fully specified at the start of the project— something that is not always possible. Plus, projects do not always proceed in a linear fashion: requirements change, staff come

and go, outside events may cause delays. All these call for some form of iteration in the lifecycle, which the waterfall process model caters for only nominally. However, this process works well if the product requirements are well understood, where quality requirements dominate cost and schedule requirements, and where the team is inexperienced or technically weak as the structure helps to minimise wasted effort (McConnell, 1996, p. 137).

Other process models, such as Scrum, deliver software in small, working increments. Users have a chance to see and use the software early in the project, which helps them refine their requirements and it also highlights any business or technical risks while there is still time and budget to take corrective action. However, agile development processes do not yet scale well, being more suited to projects lasting up to 12 months and teams of perhaps to 10 or so members (Ambler, 2006, p. 46). Also, it may be difficult to convince project sponsors that the process can be controlled effectively and therefore agile projects demand greater risk management and control skills (Boehm & Turner, 2003).

So, the development process chosen for a particular project needs to take into account the development maturity of the team, the nature and scale of the project being undertaken, and constraints such as time and budget.

Some argue that the heavyweight/lightweight continuum is not enough (Highsmith, 2000, pp. 4 – 8; Meso & Jain, 2006, p. 20), and to accurately describe the way software is developed we need to borrow a concept that is usually applied to biological and sociological models: complex adaptive systems. A complex adaptive system is one containing a group of self-similar agents who participate in a rich exchange of information or materials with each other according to some simple rules, and thereby learn and adapt to conditions around them (Buckley, 1968; Dooley, 1996; Anderson, 1999). When the principles of complex adaptive systems are mapped against agile development practices (Table 2), we can see that there is a tolerable correspondence.

Taking a complex adaptive systems viewpoint is not just an interesting thought experiment—it has been used to design and develop enterprise application integration projects in health care (Tan et al., 2005) and the insurance industry (Sutherland & van den Heuvel, 2002). Complex adaptive systems also have

implications for the way software is developed because we may have been on the wrong track:

Blame for the systems development crisis has been laid at the feet of the

Table 2: The principles of complex adaptive systems mapped against agile development practices (Meso & Jain, 2006, p. 23).

Agile Practice	Complex Adaptive System Principle	Example
Frequent releases and continuous integration	Principle of growth and evolution	Software is developed iterative and incrementally, gradually adding more and more features onto a working base product.
Obtain frequent feedback from the end users	Principle of transformative feedback loops	Development cycles are time-boxed, after which the software is demonstrated and feedback is gathered from the end users.
Accommodate changes in the requirements	Principle of emergent order	Working software is demonstrated regularly so that requirements can be gathered and refined. Work items in the product backlog are reworked and re-prioritised as needed.
Loosely coupled development environment	Principle of distributed control	Software is developed as loosely coupled components which communicate through defined interfaces. Decision-making authority is pushed down the management hierarchy closer to those performing the actions.
Planning is kept to a minimum	Principle of growth and evolution Principle of emergent order	Enough architectural and project planning is done to satisfy immediately foreseeable needs, which is based on the understanding that detailed long-term planning is wasted effort when change is continuous.
Promote continuous learning and improvement	Principle of growth and evolution Principle of interaction and relationships	Allow for manageable experimentation in product design and implementation so that lessons can be learned.
Emphasis on working software	Principle of path of least effort	At the daily standup meetings developers can raise issues that are blocking their progress. These should be dealt with as soon as possible.

creators of development methods, tool builders, analysts, designers and implementers. But we suggest that the problem may, instead, lie in an incorrect goal set that we all have accepted from the outset [and] that is the idea that systems should support organizational stability and structure, should be low maintenance, and should strive for high degrees of user acceptance. (Truex et al., 1999, pp. 122 – 123)

Truex and colleagues believe that the accepted goals don't match the way organisations really operate: stable and low maintenance systems discourage even positive strategic change because of the presumed cost and for fear of disrupting something now working well; and user acceptance is improbable because they may not fully understand what they need. Instead, the authors say software development needs to accommodate continuous analysis (because change in the surrounding business and technical environment is constant); continuous requirements negotiations (because there will always be a conflict between what the users have and what they need); and a portfolio of ongoing maintenance activities (because systems should never be allowed to become totally outdated or irreparable).

So, in just one of the accidental attributes of software— the process by which it is created and maintained— software engineers have plenty of choices; not so plentiful are the means by which to make a wise discrimination.

Preparing Software Engineers for a Complex Environment

How then do we prepare software engineers to work in an environment that is complex in and of itself, and which is, in turn, used to create a complex product? To answer this we need to look at the state of current professional practice and the educational programs that produce new software engineers.

The Profession of Software Engineering

In response to the natural characteristics of software and the way it is developed, practitioners have sought ways of making software development more reliable, more predictable— more like its engineering namesake. Among the most common definitions of *software* engineering are:

- “... the need for software manufacture to be based on the types of theoretical foundations and practical disciplines, that are traditional in the established branches of engineering.” (Naur & Randell, 1969, p. 13)

-
- “1. The systematic application of scientific and technological knowledge, methods, and experience to the design, implementation, testing, and documentation of software... 2. The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software” (ISO/IEC/IEEE, 2010, p. 331)
 - “Software engineering is an engineering discipline that is concerned with all aspects of software production from the early stages of system specification to maintaining the system after it has gone into use... In general software engineers adopt a systematic and organised approach to their work, as this is often the most effective way to produce high-quality software” (Sommerville, 2007, p. 7)

These definitions stress that software development needs the rigorous foundation that an engineering process can provide, underpinned by mathematics and science (Bryant, 2000, p. 78). As used here, an engineering process is considered to be a “repeated cycle through requirements, specifications, prototypes, and testing” (Denning & Riehle, 2009, pp. 24 - 25), meaning it covers a broad spectrum of technical and project management activities and not just the design and coding of software. But, studies have found that software engineers do not consistently practice sound engineering principles such as separating design from implementation, collecting and analysing metrics, and striving for predictable outcomes (the principle of least surprise) (Humphrey, 1998; Riehle, 2008; Denning & Riehle, 2009).

Even though the engineering metaphor tends to break down on close examination, and others have proposed craftsmanship (McBreen, 2001; Martin, 2009) and even gardening (Hunt & Thomas, 1999; Buschmann, 2011a, 2011b), the title first used at those NATO conferences in 1968 and 1969 has stuck fast. It may be more helpful to view software engineering as a profession, for which there are concrete definitions and parameters. For example:

The legitimization of professional authority involves three distinctive claims: first, that the knowledge and competence of the professional have been validated by a community of his or her peers; second, that this consensually validated knowledge rests on rational, scientific grounds; and third, that the professional’s judgment and advice are oriented toward a set of substantive values, such as health. These aspects of legitimacy

correspond to the kinds of attributes—collegial, cognitive, and moral—usually embodied in the term “profession.” (Starr, 1984, p. 15)

By analysing a range of recognised professions in light of this definition, Ford and Gibbs (1996) were able to say what characterises a profession and how these components are related to each other (Figure 5).

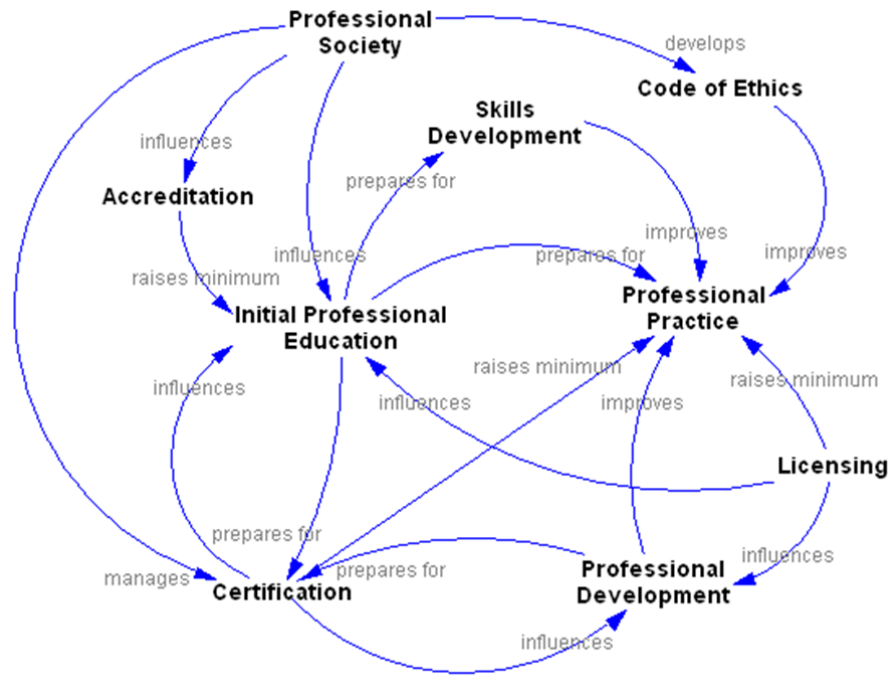


Figure 5: Causal relationships amongst the components of recognised professions

What is evident from Figure 5 is the central role of professional initial and ongoing education: it underpins many of the other components and so is crucial to the development of competent and proficient software engineers and crucial in addressing the quality and productivity issues of the field.

Table 3 shows that when we map software engineering against Ford and Gibbs’ professional components, the field is still relatively immature, but there has been a significant improvement from 1996 when the first assessment was done.

Ford and Gibbs (1996, p. 8) note that only a handful of professions, such as medicine and law, are truly mature in all components and it may take many, many years to reach this stage. The immature, but improving, state of the software engineering professional is therefore a long-term project.

Table 3: Software engineering mapped against Ford & Gibbs (1996) components of professional practice

Component	1996 ^{1,2}	2003 ^{2,3}	2011
<p>Initial professional practice: this means knowledge of an advanced type usually acquired over a prolonged course of specialised intellectual instruction and study (Ford & Gibbs, 1996, p. 10), such as an undergraduate or post-graduate university degree.</p>	Ad-hoc	Between ad-hoc and specific	<p>Maturing. In Australia, for example, there are 14 institutions offering 18 specialised undergraduate and post-graduate degrees in software engineering. (Hobsons, 2011)</p>
<p>Accreditation: a way of assuring the quality of education programs. Recognised and independent bodies accredit courses to say they meet the standards of the profession (Ford & Gibbs, 1996, p. 11).</p>	Ad-hoc	Specific	<p>Specific. Software engineering accrediting bodies exist in Australia, Canada, Japan, the United Kingdom and the United States (Frezza et al., 2006).</p>
<p>Skills development: professionals are expected to learn their field's body of knowledge and also develop their skills in the application of that knowledge. In the past, practical skills were developed through apprenticeships; now, they are more likely to be developed through laboratory exercises, student projects, or internships (Ford & Gibbs, 1996, pp. 12-13).</p>	Ad-hoc	Specific	<p>Specific. The SE2004 curriculum guidelines define the knowledge a software engineer must know upon graduation, and the Software Engineering Body of Knowledge (SWEBOK) (Bourque et al., 1999) defines the body of knowledge a software engineer should have after four years of practice. However, but there is no accepted path between the two.</p>
<p>Certification: members of a profession can voluntarily apply to be certified by a professional society to say that they have achieved a certain competency (Ford & Gibbs, 1996, pp. 13-14)</p>	Ad-hoc	Specific	<p>Specific. The IEEE's Certified Software Development Professional program is based on the SWEBOK and is an independent measure by which members of the field can be assessed (Naveda & Seidman, 2005). Commercial vendors such as Microsoft, Cisco, and Oracle also provide technology-specific certification programs.</p>
<p>Licensing: a mandatory</p>	Ad-hoc	Ad-hoc	<p>Specific. While some states in</p>

<p>process, usually administered by a government agency, which confirms a person is knowledgeable and competent in their field.</p>			<p>the US and provinces in Canada started licensing software engineers in 1998 and 1999 respectively, the practice has not been widely adopted because the onerous conditions mean only a tiny fraction would qualify or even bother (Melody, 2003; McConnell, 2004, p. 56)</p>
<p>Professional development: covers those things a person does to maintain their skills and knowledge after they start practice. "It includes everything from the occasional reading of an article in a professional magazine to lengthy continuing education or training required for relicensing or recertification" (Ford & Gibbs, 1996, p. 15).</p>	<p>Specific</p>	<p>Ad-hoc moving towards Specific</p>	<p>Specific. "Professional development is perhaps the most nebulous of the eight infrastructure components, in that it includes many kinds of activities, each of which can take many forms" (Ford & Gibbs, 1996, p. 33). While there are many training and further education options for software engineers, the path they take is less planned and more dictated by their current work assignment and their available resources.</p>
<p>Code of ethics: a code of conduct or practice that ensures practitioners behave in an ethical and responsible way. Committing to a code of ethics make practitioners feel part of a community and, if sanctions are given for violations, people dealing with practitioners know they have recourse if something goes wrong. (Ford & Gibbs, 1996, pp. 16-17)</p>	<p>Ad-hoc</p>	<p>Specific</p>	<p>Specific. The IEEE and ACM have created a code of ethics for software engineers, but this hasn't been widely adopted by industry or academia and isn't enforceable (McConnell, 2004, p. 57).</p>
<p>Professional society: a body that promotes the exchange of professional knowledge by, say, publishing journals and magazines, organising conferences and symposia, and publishing text or reference books. Professional societies will also typically be involved in defining certification standards, managing accreditation programs, and</p>	<p>Specific</p>	<p>Specific</p>	<p>Maturing. Two main professional societies, the IEEE and the ACM, each have specialised software engineering divisions, publications, and conferences and they also cooperate on curriculum, accreditation, and certification development.</p>

creating academic curriculum
guidelines.

1. (Ford & Gibbs, 1996)
2. **Ad-hoc**: some related form of the component exists, but it is not specifically related to the given profession. **Specific**: the component exists and is clearly identified with the given profession. **Maturing**: the component has existed for many years, during which time it has come under active stewardship of an appropriate body with the profession and is being continually improved.
3. (McConnell, 2004)

The Education of Software Engineers

As mentioned in the previous section, the initial and ongoing professional education of software engineers plays a critical role in the field's ability to tackle the natural complexity of software and how it is developed. In recognition of this, the two main professional bodies, the IEEE and ACM, have for some time jointly developed a range of curriculum guidelines to help educators create comprehensive and relevant courses, and to form the basis of certification and accreditation programs. Reflecting the breadth and depth of computer science as it now stands, these curriculum guidelines span a number of volumes:

- Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering (SE2004)
- Curriculum Guidelines for Graduate Degree Programs in Software Engineering (GSWE2009)
- Curriculum Guidelines for Undergraduate Degree Programs in Information Systems (IS2010)
- Model Curriculum and Guidelines for Graduate Degree Programs in Information Systems (MSIS2006)
- Curriculum Guidelines for Undergraduate Degree Programs in Computer Engineering (CE 2004)
- The Computing Curricula Information Technology Volume (IT2008)

Of interest here are SE2004, GSWE2009, IS2010, and MSIS2006 because they directly address aspects of software engineering management.

For example, IS2010 places great emphasis on leadership, collaboration, negotiation, and ethical behaviour because, "it is impossible for IS graduates to exhibit the required high-level IS capabilities without these foundation knowledge and skills" (Joint IS2010 Curriculum Task Force, 2010, p. 21). The recommended

course, IS2010.5 IS Project Management, is designed to teach students the methods, techniques, and tools that organisations use to manage their information systems projects. However, “the course specification intentionally leaves discussion regarding specific methods and approaches unanswered” (Joint IS2010 Curriculum Task Force, 2010, p. 50), which means institutions need to figure out for themselves how best to teach these aspects.

Similarly, SE2004, which is explicitly based on the SWEBOK, says students must be exposed to:

... aspects of professional practice as an integral component of the undergraduate curriculum. The professional practice of software engineering encompasses a wide range of issues and activities, including problem solving, management, ethical and legal concerns, written and oral communication, working as part of a team, and remaining current in a rapidly changing discipline. (Joint Task Force on Computing Curriculum, 2004, p. 15; see also Lethbridge et al., 2006; Hazzan, 2010)

To achieve these outcomes, SE2004 defines nine Software Engineering Education Knowledge (SEEK) areas, each of which has a series of associated knowledge units. For all knowledge areas and units, Bloom (1956) attributes are specified as either: knowledge, comprehension or application. To briefly recap, the Bloom taxonomy is a classification of learning objectives consisting of three domains: cognitive, affective and psychomotor. The cognitive domain, which is most applicable to traditional teaching, defines six levels of taxonomy from the lowest to the highest:

- Knowledge: remember previously-learned materials by recalling specific facts, terminology, theories and answers.
- Comprehension: demonstrate an understanding of information by being able to compare, contrast, organize, interpret, describe, and extrapolate.
- Application: use previously-learned material in new situations.
- Analysis: decompose previously-learned material into parts in order find patterns and to make inferences and generalisations.
- Synthesis: use existing ideas in different ways to create new ideas or to propose alternative solutions.
- Evaluation: judge the validity of ideas or information within a certain context.

(There is some debate in education circles about whether synthesis or evaluation is the higher Bloom level. Here, the most commonly published order has been used.)

Key amongst the SE2004 knowledge areas is Software Management (MGT), which represents approximately 4 per cent of the taught-load component, and is made up of five knowledge units:

- Management concepts
- Project planning
- Project personnel and organisation
- Project control
- Software configuration and management

The knowledge units Project Planning and Project Personnel and Organization are each given the Bloom classification level of Application (Table 4).

Table 4: SE2004 Project Planning and Project Personnel and Organization topics along with their Bloom (1956) classifications

SE2004 Project Planning topics		
KA/KU	Topic	Bloom's taxonomy
MGT.pp	Project planning	
MGT.pp.1	Evaluation and planning	Comprehension
MGT.pp.2	Work breakdown structure	Application
MGT.pp.3	Task scheduling	Application
MGT.pp.4	Effort estimation	Application
MGT.pp.5	Resource allocation	Comprehension
MGT.pp.6	Risk management	Application
SE2004 Project Personnel and Organization topics		
KA/KU	Topic	Bloom's taxonomy
MGT.per	Project personnel and organization	
MGT.per.1	Organizational structures, positions, responsibilities and authority	Knowledge
MGT.per.2	Formal/informal communication	Knowledge
MGT.per.3	Project staffing	Knowledge
MGT.per.4	Personnel training, career development, and evaluation	Knowledge
MGT.per.5	Meeting management	Application
MGT.per.6	Building and motivating teams	Application
MGT.per.7	Conflict resolution	Application

To teach these, SE2004's curriculum guideline 17 (Joint Task Force on Computing Curriculum, 2004, p. 45) encourages a variety of teaching and learning methods that include problem-based learning, just-in-time learning, learning by failure and self-study.

In a similar way to IS2010 and SE2004, the GSWE2009 defines a Core Body of Knowledge (CBOK) along with associated Bloom classifications. GSWE2009 recommends the 7 to 9 percent of the curriculum be devoted to software engineering management (iSSEc Project, 2009, p. 46). The distinction between GSWE2009 and SE2004 is that the former takes more units to a higher Bloom taxonomy level:

SE2004 recommends mastery of many topics at level 1. Every topic in GSWE2009 must be mastered at level 2 or higher. Moreover, many more topics in GSWE2009 require mastery at level 3 than does SE2004; e.g., in SE2004, the topic of software process is addressed only at levels 1 and 2. In GSWE2009, the same topic is covered at levels 2 and 3. (iSSEc Project, 2009, p. 15)

Software engineering management is also a key part of MSIS2006, where it is demonstrated most clearly in the practicum, which is:

... a term-long project solving a real problem for a real client against a time deadline. For full-time students, it is recommended that they work in teams and that industry supports the project by providing stipends to the students for their work because the financial incentive has been shown to improve the relevance of the project topic and the quality of the student output. For parttime, working students, a project for their employer is usually appropriate as a practicum. (MSIS2006)

MSIS2006 recognises it may not be possible to provide a practicum for all students because of cost and simple logistics and a normal capstone project is the default.

Although detailed and comprehensive, SE2004, IS2010, MSIS2006, and GSWE2009 leave some gaps. While they encourage implementers to use a variety of teaching methods, the course specifications are often intentionally vague, particularly around some social, non-technical skills, which means institutions need to figure out for themselves how best to teach these aspects.

Software Engineering in a Social Environment

It is surprising that the curriculum guidelines provide little guidance in how to teach some of the social aspects of software engineering since these have been identified many times as at least equally important to the success of software projects as the technical (DeMarco, 1991; Yourdon, 1992; Constantine, 1995; Weinberg, 1998; Yourdon, 1998; DeMarco & Lister, 1999; Yourdon, 2004). Social aspects of software engineering include topics such as (Ardis et al., 2008, pp. F3H-1):

- Observing: listening, watching, and recording the behaviour of peers or clients.
- Reviewing: reading and providing feedback on documents or source code.
- Presenting: preparing and presenting information to audiences of peers and non-technical people.
- Writing: preparing written documents and other artefacts such as source code documentation.
- Planning: organising, estimating, and synthesising process activities.
- Cooperating: working together with others to complete a task.
- Reflecting: recording and learning from past events, updating plans, looking for opportunities to reuse designs or other artefacts.
- Judging: making ethical judgments, dealing with conflict, and making performance appraisals of peers.

The reasons why these topics are not addressed more fully in software engineering programs may be two-fold: the seeming arbitrariness of the sociological factors in software development is at odds with the formal and familiar technical aspects; and the lack of suitable tools with which to model and understand human dynamics.

Successful software engineering management also depends on accepting that in any social environment, such as a software development team, sensible decisions can result in counter-intuitive, and possibly counter-productive, outcomes. Consider, for example, Brooks' Law from *The Mythical Man Month* (Brooks, 1995). The title refers to that fundamental unit of measurement and scheduling, the man-month; a unit that Brooks believes is often misunderstood:

Cost does indeed vary as the product of the number of men and the number of months. Progress does not. Hence the man-month as a unit for

measuring the size of a job is a dangerous and deceptive myth. It implies that men and months are interchangeable. (Brooks, 1995, p. 16)

Because of this lack of interchangeability, Brooks' informal law states that adding more developers to a late software project in the hope of meeting a looming deadline will only make matters worse. The reason lies in the fact that software projects often cannot be broken into isolated, independent units of work, meaning that the developers need to coordinate their activities at a detailed level. Therein lies an unappreciated communications overhead. For example, if a group of n developers need to coordinate their efforts with each other then the number of communication paths can be represented by $n(n-1)/2$. Time spent navigating these paths is time not spent being directly productive.

When new developers are added to the equation, the communications overhead is amplified. The new developers are usually not immediately productive because they need to become acquainted with the overall aims of the project, its strategy and the general plan of work (Sengupta et al., 1999; Bradley & McGrath, 2000), and they possibly need to undergo some form of organisational socialisation (Schein, 1980). The best, and often only, people able to provide this training and socialisation are the existing developers, who are in the process diverted from their primary tasks.

The net result is that more time is lost in bringing the new developers up to speed and in additional coordination efforts than is gained in productive time (see Caulfield et al., 2004 for a worked example).

What are educators to make of all this? Software engineering management is a dynamic and sometimes counter-intuitive socio-technical activity which produces a complex artefact that is often difficult to reason about and test. In some critical areas, current curriculum guidelines leave it to educators to bridge the pedagogical gap between the qualities and skills a well-rounded software engineer should possess and how these should be taught.

This gap is exposed most often when students finish their requisite courses and attempt their final, important, and synthesising capstone project. While there are many cases of capstone projects bringing great benefits for the students and their clients (Boehm et al., 1998; Johns-Boast & Patch, 2010), these are balanced by

stories of significant failures in which student/client relationships broke down, there was severe internal team dissension, or the final software was unusable (Brereton et al., 2000; Polack-Wahl, 2006; Cheng & Lin, 2010). For those capstone projects that failed, there was little opportunity for reflection or remedial action because the project was the final unit of study for the course. Some research has been conducted that recommends guidelines for successful capstone projects (Robillard & Robillard, 1998), such as providing students with basic training in project control, reviewing the design documents, and having an experienced software engineer mentor certain stages, but these are relatively costly or time-consuming course attributes and there is little evidence they have been widely adopted.

One possible solution to the problems presented in this section is to look at how we might deal with complexity and change more generally and see if there are lessons that can be applied for software engineering management.

Dealing With Complexity and Change

In 1979, *The Learning Report* (Botkin et al.) was presented at a Club of Rome conference—a conference of leading business, scientific, social, and political thinkers meeting with a view of finding holistic and interdisciplinary solutions to intractable problems. The report was the culmination of two years of meetings, seminars, and discussions concerning the world problematique: a snarl of problems in areas such as energy, population, and food, often with political, social, and psychological aspects, the outcome of which was a degree of unparalleled world complexity. The report saw a growing gap between a complexity of human making and a lagging development of our own capacities to deal with it, and proposed a means of bridging this gap—anticipatory learning. Anticipatory learning offers a possible solution by being:

- *Future-oriented.* It assumes an orientation that is not solely reactive and prepares for possible contingencies and considers long-range future alternatives. “Through anticipatory learning, the future may enter our lives as a friend, not as a burglar” (Botkin et al., 1979, p. 13).
- *Participative.* Anticipatory learning is not possible while there is a paternalistic assumption that one group has all the answers, and will deliver these to a less-informed constituency. When issues are explored as a joint venture then

solutions become “almost self-evident, are better supported, can be more readily implemented, and are less likely to generate unwanted repercussions” (Botkin et al., 1979, p. 30).

The world problematique that lies behind anticipatory learning reveals itself in many ways. For example:

Perhaps for the first time in history, humankind has the capacity to create far more information than anyone can absorb, to foster far greater interdependency than anyone can manage, and to accelerate change far faster than anyone’s ability to keep pace. (Senge, 1990, p. 69)

Along the same lines, Boulding has said:

As far as many statistical series related to activities of mankind are concerned, the date that divides human history into two equal parts is well within living memory... In a very real sense the changes in the state of mankind since the date of my birth [1910] have been greater than the changes that took place in many thousands of years before this date. (Boulding, 1964, pp. 7, 8)

Meanwhile, Toffler (1970, p. 16) has noted that it is still within living memory that agriculture, the original basis of civilisation, has lost its dominance as the primary employer of the economically active population.

The same is true in software engineering management:

[The] velocity, or the rate at which business processes occur, becomes a crucial measure for enterprises competing in the fast-changing and unpredictable markets. To increase the velocity of an enterprise, its supporting information systems must be capable of rapid-changes in lock-step with business changes. Unfortunately, this rapid change is rarely possible today. (Yeh, 2002, p. 2)

If we accept that change-induced, cascading complexity is happening, then we should also accept that we need to deal with its consequences in some manner. Typically, this can be done in two ways:

- Simplify reality. Look for the primitives and hierarchy of the problem domain. Seeing abstractions or commonalities and understanding how they relate to

each other can help orient our thinking when confronted with something unfamiliar (von Bertalanffy, 1968; Courtois, 1985).

- Absorb the complexity or achieve a level of requisite variety (Ashby, 1956). When we are confronted with some new situation, piece of information, it is usually compared to an array of previous knowledge, our mental models. An inference process then tries to make sense of the new information by relating it to what is already known.

Taking these steps are the initial actions in analysing something new; we then need to decide what to do based on what we understand. Yet, in order for the later inference process to have any substance with which to work, our store of mental models, or contexts, must be sufficient:

In order to enhance the human capacity to act in new situations and to deal with unfamiliar events, innovative [anticipatory] learning requires the absorption of vast collections of contexts. When contexts are restricted, the probability of shock learning increases, for shock may be conceived as a sudden event that occurs outside the known contexts. Hence one task of innovative learning is to enhance the individual's ability to find, absorb, and create new contexts— in short, to enrich the supply of contexts. (Botkin et al., 1979, p. 24)

One way to enrich the supply of contexts is to use a tool of anticipatory learning— games (Fulmer, 1993; Senge & Fulmer, 1993). The argument is illustrated by the feedback diagram in the Figure 6 (Sterman, 2000, p. 34).

We receive information in its many forms from the real world in which we live. Based on this feedback, we make decisions that are filtered through our existing mental models, in the process changing or confirming the structure of our real-world systems and creating new decision rules and new strategies or reinforcing the existing. Games act as an alternative to applying our decisions to the real-world, a way of quickly and inexpensively experimenting with different policies and thereby increasing our supply of contexts. Without the tool of simulation, we must directly respond to real-world feedback that is “very slow and often rendered ineffective by dynamic complexity, time delays, inadequate and ambiguous feedback, poor reasoning skills, defensive reactions, and the costs of experimentation” (Sterman, 2000, p. 37).

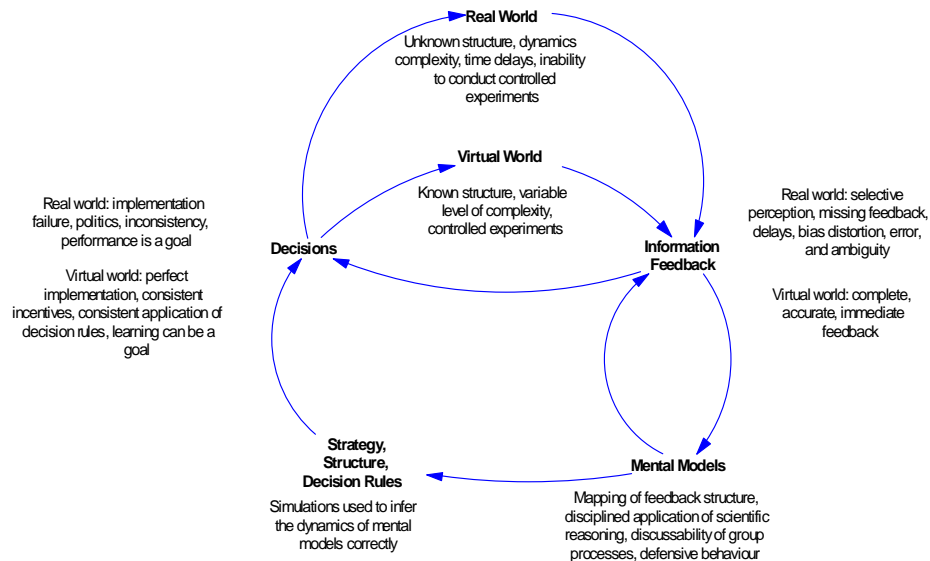


Figure 6: Idealised learning process

Games could therefore be one way of preparing students for a complex and dynamic working environment such as software engineering management. Before seeing how this might be so, we must look at problem-based learning—the broader educational theory of which anticipatory learning is a subset—because it provides a sound intellectual framework upon which to build practical implementations such as games.

Problem-Based Learning

Introducing Problem-Based Learning

Problem-based learning is a pedagogic method that uses problem scenarios to encourage students to work out solutions for themselves. Usually working in small teams, students explore the problem, bring their personal experience to bear, identify any gaps in their knowledge, and eventually come up with viable solutions. The problems themselves are usually complex, ill-defined, real-world situations for which there may not necessarily be a single right or wrong solution (Maxwell et al., 2004, p. 2). The students build new knowledge through self-directed learning while their tutors act as facilitators or consultants rather than more traditional instructors (Dempsey et al., 2002, p. 5; Woolfolk, 2009, p. 347; McCall, 2011).

At first glance, problem-based learning may look like other forms of active learning, however Table 5 shows there are some unique characteristics (Levin et al., 2001, p. 123; Savin-Baden & Major, 2004, p. 7; Woolfolk, 2009, pp. 348 –

349). Most noticeably, students are presented with the problems *before* they receive more traditional instruction such as lectures, tutorials, readings and assignments. This forces them to name what they need to learn to solve the problem (Wenzler, 2009, p. 59) and to explore the body of knowledge of their field with a practical goal in mind.

Philosophical Under-Pinnings

Problem-based learning has its origins in a number of schools of philosophical and

Table 5: Problem-based learning compared to other active learning methods.

Method	Organisation of knowledge	Forms of knowledge	Role of student	Role of tutor	Type of activity
Problem-based learning	Open-ended situations and problems. These are presented <i>before</i> formal instruction.	Contingent and constructed	Active participants and independent critical inquirers	A facilitator or consultant	Development of strategies to help team and individual learning
Project-based learning	Tutor-set, structured tasks	Performative and practical	Completer of project or member of a project team that develops a solution or strategy	Task setter and project supervisor	Problem solving and problem management
Problem-solving learning	Step-by-step logical problem solving through knowledge supplied by the tutor	Largely propositional but may also be practical	Problem solver who acquires knowledge through bounded problem solving	A guide to the right knowledge and solution	Finding solutions to given problems
Action learning	Group-led discussion and reflection on actions	Personal and performative	Self-advisor who seeks to achieve their goals and help others achieve theirs through reflection and action	A facilitator of reflection and action	Achievement of individual goals

educational thought that have meditated on the way we acquire or develop knowledge:

- **Metaphysical critiquing of knowledge through reason:** problem-based learning assumes learners will develop meta-cognitive skills and therefore will use reasoning abilities to solve complex problems (Hacker & Dunlosky, 2003; Savin-Baden & Major, 2004, p. 11). These ideas date back to the fifth- and fourth-century BC Greek philosophers, such as Socrates, Plato, and Aristotle. Aristotle, for example, believed that knowledge could be gained through perception as well as through abstraction and logical reasoning. To this end, he trained his students to use Socratic dialectic to reconcile the contradictions presented in a thesis or problem. He saw a liberal and technical education on many subjects and theories as a way for students to make rational judgments on almost any subject. In the same way, problem-based learning encourages students towards sustained enquiry leading to practical skills.
- **Deductive and inductive reasoning:** in problem-based learning, students assume no givens and must use deductive and inductive reasoning and experimentation to test the validity of a particular course of action.
- **Positivism and social justice:** problem-based learning presents an opportunity to address many issues of social justice as it provides an opportunity for students who might otherwise be marginalised in traditional classes to participate. The small-team structure of a problem-based learning class means there is less chance that students who are traditionally marginalised (the poor, the socially inept, students whose first language is not English, for example) will be excluded.
- **Existentialism and independent thinking:** one of the primary goals of problem-based learning is to help students develop independent learning skills. Existentialist philosophers such as Søren Kierkegaard (1813 – 1855) and Friedrich Nietzsche (1844 – 1900) saw this. Kierkegaard believed that individuals learn by observing others and experimenting rather than being told information (Kriz, 2009, p. 28). Meanwhile, Nietzsche believed in learning to think and criticised schools and universities for no longer understanding this. He questioned the value of educators as he believed that one could not educate another and that education must necessarily be self-education otherwise it becomes a form of control and levelling. While not going to a Nietzschean extreme, problem-based learning encourages students to think critically, and to take responsibility for their own learning and for that of others in their team.

Background and History

From these philosophical foundations, problem-based learning developed a practical realisation, first in medical schools in North America before spreading to other places and disciplines.

Medical Schools

By the late 19th century, North American medical schools were mostly privately-funded, poorly-run institutions that produced an over-abundance of physicians of greatly-varying skills (Flexner, 1910). Flexner's report saw that medical training had to be held to higher standards and called for university-based academic and clinical training closely tied with mainstream science. Most of Flexner's recommendations were adopted and within 25 years more than half the medical schools in America had closed or merged and most of those that remained were affiliated with a university (Savin-Baden & Major, 2004, p. 16).

By the 1960s, the Flexner model of medical education had developed problems of its own: it was found that students lacked critical-thinking and problem-solving skills because the structure of the curriculum and examinations rewarded rote learning. Students were memorising propositional knowledge without understanding how to apply it (Levin & Forman, 1973, p. 867; Savin-Baden & Major, 2004, p. 17).

In 1966, an opportunity arose to address these latest problems. A new hospital and medical school were being planned in Ontario, Canada, affiliated with McMaster University. As a green-field development, the school had no pre-existing curriculum so the academic staff envisaged a different way of doing things, a way that presented students with patient problems similar to the way practising physicians might have encountered them. Without formal lectures, students, working in small teams, were given a 'problem pack'— a deck of structured cards that described a patient's problem— that they had to research and then come up with a viable solution (Barrows & Tamblyn, 1977). Compared with students in a control group taking a more traditional learning path, those using the problem-based learning format (as it became known) were found to be more motivated, had better problem-solving skills, and were better able to search for the solution to their problem (Barrows & Tamblyn, 1976).

Based on the initial success of the McMaster model, medical schools at the University of Limburg at Maastricht in the Netherlands and the University of Newcastle in Australia implemented their own problem-based learning curriculums during the 1970s. At other medical schools, with significant cultural and administrative investments in traditional learning models, change was slower. However, by 1985 even heavy-weight institutions such as Harvard Medical School were using problem-based learning, albeit interspersed with formal lectures. The technique has enjoyed a steady adoption and is now widely used in medical schools throughout Europe, South America, Africa, and Asia (Savery & Duffy, 1994).

Problem Based Learning in Other Fields

Problem-based learning began a seemingly natural progression to other health-related fields during the 1980s, such as pharmacy at Stanford University, nursing at the universities of North Carolina and Newcastle, and veterinary science at Mississippi State University.

With a foothold in these areas of the curriculum, problem-based learning gradually became known and adopted in areas outside of health such as business education (Garris et al., 2002), teaching science and maths (Craik & Craik, 1986; Duch et al., 2001; Ronis, 2008), teacher education (Levin, 2001), chemical engineering (Woods, 1996), as well as in the arts and humanities (Amador et al., 2007). Problem-based learning is also being used in software engineering education but usually in one-off units or in capstone projects and not as a fully-integrated part of a multi-year degree (Armarego, 2002; Levy et al., 2008; Qiu & Chen, 2010).

Criticisms of Problem-Based Learning

In general, problem-based learning takes more time and effort on the part of all participants (Levin et al., 2001, p. 129). Teachers need to prepare detailed lesson plans (Barell, 2006, pp. 52 – 54) and use a range of formative and summative assessment techniques (Anderson & Puckett, 2003; Savin-Baden & Major, 2004, pp. 118 – 119); while students need to become familiar with new demands such as coordinating their activities with a team (Amador et al., 2007, pp. 37 – 41). Efficiency practices are a mitigation, but many problem-based learning implementers find they are dealing with inherent rather than accidental qualities of the technique (Schwartz et al., 2001).

It has been said that problem-based learning is more concerned with long-term learning rather than immediate learning outcomes. Becoming a life-long learner and a critical problem-solver are noble goals, but will the student know enough to pass the end-of-semester examination? Research in the area returns a qualified, *yes* (Hung et al., 2003, pp. 15 – 17). If carefully designed, a problem-based learning course can simultaneously meet long- and short-term learning goals. But, if sufficient time and effort cannot be devoted to this design or the learning objectives are simply factual, then a more traditional method is more appropriate.

Compared to traditional instructional courses, students may find the active participation demanded by problem-based learning unfamiliar and they may flounder, thereby putting at risk the investment in their education and perhaps their long-term career prospects. To overcome this, students need to understand and accept their role as a self-directed learner along with the advantages this brings. While giving due care to any apprehensions they might have, subtlety is wasteful: the nature of the student's role must be made explicit on the first day of the course (Bereiter & Scardamalia, 1993, p. 59; Schwartz et al., 2001; Savin-Baden & Major, 2004, chapter 8).

Problem-based learning means teachers have to take on a slightly different role too: instead of transmitting knowledge and information they have to become facilitators of thinking and learning (Bridges, 1992, pp. 58 – 64; Hung et al., 2003, p. 19; Savin-Baden, 2003, pp. 35 – 47); something they might not be experienced with or comfortable doing. For example, a good facilitator needs a range of skills including reflection, dispute resolution, quickly dealing with inactive or domineering team members, and even reading body language and other non-verbal cues (Savin-Baden & Major, 2004, pp. 96 - 99). Some argue that these are skills teachers already have and they just have to be resurfaced (Savin-Baden, 2003, pp. 77 – 89), but for best effect facilitators need at least some basic training in order to make each learning session as valuable as possible (Savin-Baden, 2003, p. 24).

Is Problem-Based Learning Worth the Effort?

So, it seems that implementing problem-based learning means extra work for teachers and students, and it must compete against the inertia of an established,

proven instructional model that is already familiar to educators and students. Nonetheless, problem-based learning has some worthy advantages:

- It has a sound philosophical grounding.
- A body of literature has grown which supports its prime assumption that students are able to transfer knowledge and skills from one context (academia) to another (the real world) (Savin-Baden & Wilkie, 1996; Duch et al., 2001; Schwartz et al., 2001; Savin-Baden & Major, 2004, p. 60; Barell, 2006).
- If students fully embrace the technique they have the chance to take more control over the direction of their education rather than being passive sponges.

Accepting these advantages, problem-based learning can be implemented in many ways— from the original problem pack deck of cards used by McMaster University medical students to moot courts. Considered next is one way to implement problem-based learning: games.

Games as an Implementation of Problem-Based Learning

History and Origins

In addition to problem-based learning, games draw their intellectual integrity from a number of sources including educational theory (Dewey, 1938/1963; Lewin, 1952; Papert, 1980; Kolb, 1984), operations research (Thomas & Deemer, 1957; Wilson, 1968, pp. 36 - 50), small-group behaviour research (Kennedy, 1971b, 1971a), war-gaming, decision sciences (Mayer, 2009, p. 827), and systems engineering (Raser, 1969, pp. 46 - 55).

By way of illustration, the most dominant antecedent, war gaming, will be followed here with a view of showing that games have a maturity, breadth, and substance beyond the unprepossessing name.

An Origin in War Games

In the science fiction novel *Ender's Game* (Card, 1985), alien attacks on the Earth prompt the world government to begin training children as future military commanders (a multi-generation war is anticipated) by setting them to play continuous tactical and strategic computer war games. Amongst the current crop, one player stands out, Ender Wiggin— he wins every time. Ender is selected to

lead a group of team-mates in a complex computer battle simulation of a final confrontation with the aliens; a confrontation that ends when the aliens' home planet is destroyed. Only later is it revealed that Ender and his fellow players have been commanding real weapons and that the end of the game also means the end of the alien threat.

Perhaps this futuristic scenario represents the end-game of war games, where game and reality are inseparable. Yet, fundamentally, the antecedents of modern war games were indeed, just games.

To start, a war game can be thought of as a re-creation of the activity of war that uses war's vocabulary and which has as its ultimate goal the preparation and education of players for the realities of war before the event. "There will not be time enough after the outbreak of war for [an officer] to learn his duties before military operations begin; and the cost of permitting him to learn by experience derived from his own blunders is too great to be considered" (Sayre, 1911, p. ix).

While the exact origins of war games are somewhat unclear, students of the history of chess, and similar board games played for pleasure, have noted that at an early stage such games were used as symbolic equivalents of warfare (Murray, 1913, pp. 42 - 50). For example, in the ancient Chinese game of Wei-Hai, dating from around 3000 BC, players moved coloured stones on a grid with the goal of controlling as much space as possible (Wilson, 1968, p. 1; Smith, 2010, p. 7). While no diagrams or game pieces for Wei-Hai have survived, descriptions suggest it resembled the modern Japanese game of Go (Smith, 1998, p. 805). In the Indian game of Chaturanga (1000 BC), generally assumed to be the oldest form of chess (Murray, 1913, p. 42), a group of up to four players used a board divided into squares, and pieces in the shapes of elephants, soldiers, cavalry, and nobles. In contrast to Wei-Hai, the object was to capture the opponents' pieces rather than to control territory and chance elements are introduced by a dice.

Little evidence exists that the link between games for pleasure and the study of war persisted beyond these early examples. However, it was with the coming of the "Age of Reason, when men decided that the conduct of war, like other human pursuits, was subject to scientific laws, that games reappeared which consciously reproduced the elements of war for play" (Wilson, 1968, p. 2). These new war games drew on chess-like variants, adding verisimilitude by using pieces shaped as

soldiers, weapons of war, and royalty. Koenigspiel, or Kings Game, developed by Christopher Weikmann at Ulm, Germany in 1664 consisted of thirty pieces and a large board (Sayre, 1911, p. 5), but remained essentially chess.

Following the Franco-Prussian War of 1870 – 1871, the war games concept spread to other countries. In 1872, war games were introduced into the British Army (Wilson, 1968, pp. 7 - 11; Lane, 1995, p. 608). Meanwhile, the American McCarty Little devised a war game in 1887 that used miniature battleships on maps (Wilson, 1968, p. 14; Perla, 1990, pp. 63 - 70; Macedonia, 2002, p. 36), and other war games were used extensively at West Point at the same time (Sayre, 1911, pp. 17 - 18, 22).

As well as being vehicles for training and education, war games were used to exercise operational plans. For example, Germany's Schlieffen Plan for the invasion of France in 1914 was informed by war game findings (Ritter, 1958, pp. 39 - 48), and as early as 1929 Germany was gaming various conflicts with Poland and studying the possible international reactions (Wilson, 1968, pp. 27 - 29; Raser, 1969, pp. 47 - 48; Perla, 1990, pp. 41 - 42). Meanwhile, in Japan, war games conducted at the Total War Research Institute and the Tokyo Naval War College, both before and during World War II, allowed participants from both government and the military to experience the domestic and international factors of war (Wilson, 1968, pp. 32 - 35).

Until the 1970s, the practice of war games has largely been physical: pieces had been moved around boards, map-based manoeuvres had added a degree of realism, and different scenarios had been played out. While computers had been used for some of the behind-the-scenes processing, they hadn't become an integral part of the war games themselves.

In 1976, then-Captain Jack Thorpe was working as a research scientist in flight training at the Williams Air Force base near Phoenix, Arizona. His research was centred around improving the flight simulators used by the Air Force to initially train pilots (Sterling, 1993; Riddell, 1997). Essentially, these machines were stand-alone devices not far removed from Edwin Link's original, pre-World War II flight simulator which had itself been an amusement park ride before being adopted by the military (Macedonia, 2002, pp. 36 - 37). The simulators were also sometimes more expensive than the vehicle they emulated and ongoing running costs were

exorbitant (Fullford, 1996, p. 179). Instead, Thorpe imagined a network of cheap simulators, for aircraft and other vehicles, through which military personnel could learn group skills as well as the traditional sole-operator skills (Alluisi, 1991).

At the time the technology did not exist to implement Thorpe's plan, but when he moved to the Defense Advanced Research Projects Agency (DARPA) in the early 1980s he became aware of a continuing experiment in distributed networking known as the ARPANET, the forerunner of the Internet. The means were then at hand. The eventual outcome was SIMNET (for simulator network), an interactive network of real-time, person-in-the-loop battle engagement and war-gaming (Alluisi, 1991). SIMNET was designed from the outset to be cheap and uncomplicated— factors which meant it worked and which made it highly attractive to its sponsors.

From this starting point, the US military now spends over US\$4 billion each year on simulation training, game development, and equipment (Macedonia, 2002, p. 33; Joyce, 2005, p. 16), while the global market is expected to be in excess of US\$8.75 billion in 2011 (Visiongain, 2010). This proves to be a cost-effective spend when compared to equivalent live training (Defense Science Board, 2002; Singer, 2010, p. 95) and which has been directly attributed to reduced battlefield casualties (Kraemer & Bessemer, 1987; Hart & Sulzen, 1988; Zorpette, 1991; Sterling, 1993; Macedonia, 2005).

The Arrival of the Modern Business Game

In 1956, the American Management Association developed what is generally considered to be the first Western business game, Top Management Decision Simulation, explicitly acknowledging its direct relation to military war gaming:

In the war games conducted by the Armed Forces, command officers of the Army, Navy, and Air Force have an opportunity to practice decision making creatively in a myriad of hypothetical yet true-to-life competitive situations. Moreover, they are forced to make decisions in areas outside their own specialty; a naval communications officer, for example may play the role of a task force commander. Why then, shouldn't businessmen have the same opportunity? (Ricciardi et al. cited in Cohen & Rhenman, 1961, p. 135)

In this game teams of players managed a company that produced a single product and competed with the products of other teams. Around the same time, the RAND

Corporation developed a game called Monopologs based on the supply logistics of the US Air Force (Jackson, 1959). Other similar games quickly followed. For example, Andlingers's (1958a) Business Management Game set two or three teams of players in competition within a market in which each team had a single product. The teams had to make decisions relating to production, finance, research and development, and advertising as they managed their companies from quarter to quarter.

Up until this time, business games were largely conducted by consulting firms for the benefit of corporate decision makers and executives. However, educators were also seeing the benefits of business games. The Top Management Decision Game developed by Schreiber, was the first business simulation game used in a university class, the business policy class at University of Washington in 1957 (Kibbee et al., 1961, p. 166). From this point onwards, the use of business games in industry and academia grew rapidly. By 1961 it was estimated that about 100 business games had been developed and more than 30,000 executives had played at least one game (Kibbee et al., 1961, p. 165). Meanwhile, a survey of 90 American business schools found that only eight had not, and were not intending to, introduce business games into the curriculum in the near future (Dale & Klasson, 1962).

While business games were being used innovatively (Naylor & Gattis, 1976; Williams, 1978) and across many different business types, such as collective labour negotiations (Veglahn et al., 1978), insurance and risk management training (Schott, 1976), and international relations (Guetzkow, 1959), by the late 1960s and early 1970s business game penetration in business and academia had peaked. While the tool had gained a certain degree of saturation, there were also some validity and reliability concerns (Neuhauser, 1976). For example, a 1970s study of game use in US colleges found only a small number of rigorous validation studies, and only one suggested that learning of any significance had taken place (Greenlaw & Wyman, 1973). But, since the 1980s there has been a resurgence due in part to:

- Improvements in the symbols and software used to map and model system structure (Gredler, 1996; Prakash et al., 2009).
- New ideas have been adopted from behavioural decision theory which help to transfer policymakers' knowledge into computer models. "Behavioural decision theory can help modelers to ask better questions of policymakers, to

specify decision processes more accurately, and to capture more or (sic) policymakers' knowledge in maps and algebra" (Morecroft, 1988, p. 315).

Currently the state of business simulation games is active and growing (Burgess, 1991; McKenna, 1991; Ellington, 1995; Faria, 1998; Faria & Wellington, 2004; Aldrich, 2005, 2009).

How Games Are Used

Contemporary games usually play one of three high-level roles: for teaching or training; to explore different policies; or as vehicles in some other form of research. The first of these is by far the most common, perhaps because it has offered the most potential:

There are apparently certain aspects of games that especially facilitate learning, such as their ability to focus attention, their requirement for action rather than merely passive observation, their abstraction of simple elements from the complex confusion of reality, and the intrinsic rewards they hold for mastery. By the combination of these properties that games provide, they show remarkable consequences as devices for learning. (Coleman, 1975b, p. 130)

These games, when used in schools, universities, executive development courses, or in-house training courses usually follow certain formats:

- General, top management, or total enterprise games require the participants to make decisions at a fairly high level and which span many functional areas in industries usually not their own. "Perhaps the greatest value of general management games is that they require planners to view their company as a total system rather than as a set of separate functional areas. For this reason, universities frequently use general management games in the capstone business policy course" (Watson & Blackstone, 1989, p. 488).
- In functional games the scope is fairly narrow and focuses on a single functional or middle management area. For example, in the Beer Distribution Game (Goodwin & Franklin, 1994; Caulfield & Maj, 2007), participants can take the role of either a small retailer, a wholesaler, or brewer and make independent decisions concerning their part of the total game.

Games can also be used operationally to explore or pre-test different policy options in, for instance, domestic or international politics (Guetzkow, 1959), ecological management (Klabbers et al., 1994; Ford, 1999, p. 164), war (Alluisi, 1991; Mastaglio & Callahan, 1995), and many social sciences (Toth, 1994; Bots & van Daalen, 2007; Mayer, 2009). “The objective is to arrive at an approximate answer through repeated trials— in essence, to arrive at a higher level of insight into a process than existed previously” (Andlinger, 1958b, p. 148) and before the actual solution is implemented (Thomas & Deemer, 1957; Kibbee et al., 1961, pp. 153 - 154; Ryan, 2000). Even though operational gaming offers an opportunity to become more fully informed, there are no guarantees:

Beyond [the] difficulty of knowing when one has solved the ‘right’ problem, there is the difficulty, particularly with gaming, of knowing when one has solved any problem... In gaming, generally, there is no way of knowing with certainty when a sample of plays is both strategically and statistically adequate for a required decision. (Thomas & Deemer, 1957, p. 19).

Finally, games can also be used as vehicles for research into human and group behaviour by placing people in teams with different organisational structures, communication channels, leadership styles, or objectives and seeing how these factors influence behavioural variables such as motivation, satisfaction and performance (Kibbee et al., 1961, pp. 151 - 153). For example Charness and colleagues (2007) used games to test the theory that people who are members of a group and identify with it will behave differently from people who perceive themselves to be isolated individuals. The grouping required by the games they used, The Battle of the Sexes and The Prisoner’s Dilemma, was incidental: the saliency of the group created in the minds of the participants was what the researchers were primarily interested in.

The Rationale for Games

In many situations, a manager in charge of a team of people may need to weigh the decisions they make with the same gravity as an engineer building an aircraft or a bridge:

The work of many managers has human consequences with potential for disaster equal to malfunctioning aeroplanes, chemical plants or dykes and dams. Nevertheless, we find it perfectly acceptable to send managers into positions of responsibility to learn by experience— by trial and error. We

ask them to learn 'by experimenting with reality'. Being intelligent people, they will recognize and fear the consequences— *and learn a lot less and slower* than they would have done otherwise. (original emphasis, de Geus, 1992, p. 4)

So, for the manager there are no laws of physics as there are to help engineers, and fear of failure can actually constipate decisions and stifle the creativity of the resulting solutions. Alternatively:

If only we managers could experiment in a laboratory before executing decisions affecting real people and real lives! Cutting up a frog seems so simple compared with cutting up and downsizing an organization. Building a clay model first and then smashing it seems far more humane than actually tearing apart an organization. (Sobkowiak & LeBleu, 1994, p. 41)

A game might be said to be a physical representation of a problem space— a clay model. As such, they are places to try new ideas and to experiment with established theories (McKenney, 1962, p. 286; Feldman, 1995, p. 355); to replay these theories as many times as needed (Gee, 2007b, p. 216); places where time and space can be contracted or expanded (Raser, 1969, pp. 109 - 110); places where it is acceptable just to try different things and where more might be learned from failure than success (Booker, 1994, p. 76; Hung et al., 2009).

It has been noted that the human capacity to understand the implications of our mental models and to accurately trace through even a small number of causal relationships is fairly limited (Miller, 1956, p. 457; Simon, 1957, p. 198; Sterman, 1994; Dangerfield & Roberts, 1995). Normally, it is difficult to reason about and share a mental model because:

One is usually only vaguely aware of one's own intuitions and assumptions. Mental models shift from moment to moment. They do not fit the linear, sequential format required by language. And words are inherently ambiguous, as are the images, thoughts, and hunches the words describe. Because of all these difficulties, verbal expression more often takes the form of advocating *what* one thinks should be done than of detailing all the semi-conscious urges that determine *why* one thinks it should be done. (Meadows & Robinson, 1985, pp. 2 - 3)

A game can make this *why* more concrete. It may be wrong, but at least it is now explicit and open to challenge and improvement.

Games may also serve a dual purpose of revealing to participants their own decision-making or leadership style (Feldman, 1995, p. 355) while also achieving overt training:

Games used jointly for experimentation and training provide an effective laboratory for conducting research in organizations. Such games permit systematic investigation of behavioural phenomena under controlled conditions, thereby facilitating the drawing of general inferences, virtually impossible in “environment-rich” situations. From the organization’s viewpoint games are educationally appealing and provide strong justification for the release of scarce and costly management talent in such activities. (Moskowitz, 1973, p. 686)

The training described here does not necessarily refer to specific skills and can instead be serendipitous and the benefits emergent:

As people weave their patterns of relationality and interdependency, their similar and differing ways of perceiving and responding emerge. The collective possibilities for learning can expand far beyond what might have been learned by any one of the participants alone. (Baker et al., 1997, pp. 7 - 8)

Many studies have reported that participants often learned more from social interactions with other players than from the game itself (Cohen et al., 1960; Dill & Doppelt, 1963, pp. 36 - 38; Greenlaw & Wyman, 1973, p. 263; Lundy, 1991; van der Meij et al., 2011). Meanwhile, other researchers (Boocock, 1970; Petranek, 1994; Prensky, 2006, pp. 106 – 108) have found that games tend to have an ice-breaking capacity among participants and open up dynamic participation.

Therefore, games have the potential to teach more than they mean.

The Instructional Value of Games

The previous section discussed the theoretical rationale for games, which was generally positive. It is naturally prudent to see what empirical evidence supports this view. Appendix N details review studies published between 1966 and the present that summarised the results of research projects into the instructional value of games. Many of the individual projects turned out to be non-experimental and descriptive, so more rigorous research is needed in the future. Nevertheless, two themes emerged.

First, learning objectives need to be clearly defined and built into the game (Hermann, 1967; Watson & Blackstone, 1989, p. 491; Becker, 2011), otherwise:

A game without [learning objectives] is like a motor without a boat— it makes a great deal of noise but doesn't get anywhere. Regardless of the size of the splash, nothing is left when the waves of excitement damp out. (Kibbee et al., 1961, p. 49)

The second theme that emerged was that games were usually able to capture the attention of players and draw them into the learning environment. But, this is just the first step and the active participation of the instructor is vital:

Early research in business gaming and experiential learning destroyed the notion that games are self teaching. Instructor guidance is critical and must be applied during crucial states in the game development to insure that learning closure takes place. Students must be guided, prompted, motivated, and sometimes forced to learn from experiences. (Knotts & Keys, 1997, p. 387)

The above comment points to a common dilemma: to a wide extent, games have been found to be more expensive and administratively demanding to develop and use than some other forms of instruction (Cohen & Rhenman, 1961, p. 151; Kibbee et al., 1961, p. 9; Babb et al., 1966, p. 471; Abt, 1970, pp. 110 - 111; Watson & Blackstone, 1989, p. 493; Petranek, 1994; Gredler, 2004). For example:

More, not less, teaching skill and preparation is required to teach a strategic management course using games and other techniques, compared with cases and text alone. Often, when a game-based strategy course has failed, the game has not failed; rather the instructor has failed to master the course elements. (Knotts & Keys, 1997, p. 392)

It must also be remembered that games are just... *games*, and as such are just one representation of how the world works. Therefore, "it is potentially dangerous to have players leave the gaming environment with the belief that the strategies that were effectively employed in playing the game are directly transferable to the real world" (Watson & Blackstone, 1989, p. 493). Participants should ideally be provided with more information than just the game to help them wisely discriminate between what may or may not work outside the game itself (Andlinger, 1958b, pp. 152 - 158).

Further, the favoured contemporary medium for games, computers, make it possible to implement games of incredible richness. Such games might be able to accommodate elements of time pressure, role-playing, feedback thought, decision-making, computer skills, random events, analysis and negotiation skills— all presented through a multi-media interface. In this situation, participants might also tend to play the game to win, as they might an arcade game, rather than to learn (Kibbee et al., 1961, p. 89; Moore, 1967, p. 22; Parasuraman, 1981). Alternatively, the richness or over-design of the game may overwhelm or discourage other participants (Andlinger, 1958b, p. 158; Abt, 1970, p. 117; Hays, 1989, p. 51). Therefore, there needs to be an appropriate emphasis on technology and a balance of game elements at each stage (Meadows, 1989, p. 639).

For best effect, games also need to be accompanied by an appropriate level of pre-game briefing and post-game debriefing (Abt, 1970, p. 116; Petranek et al., 1992; Randel et al., 1992, p. 271; Baker et al., 1997; Ryan, 2000, pp. 362 - 363; Crookall, 2010, p. 907). Some games are deliberately vague regarding the details they supply to participants, forcing the players to work out issues as part of the experience. The post-game debrief is perhaps the most critical learning component since it is here that participants can be helped to create a comparison between the game experience and their own mental models (Simons, 1993, p. 136). “The debrief can be very important in helping people to reflect on what they have experienced, in enabling them to share and debate experiences, feelings and views, and finally, in helping them to construct their experiences into understanding which can be re-applied” (Lane, 1995, p. 616).

As might be expected, the review studies in the Appendix N show that games are not orders of magnitude better (nor worse) than other pedagogical methods. Also, games can sometimes demand great effort from instructors before, during, and after game sessions to make sure players get the most from the experience. But, for this effort there are compensations such as the satisfaction of self-discovered knowledge, a richer and more varied learning experience, and the ability to rewind the play and try again without hurt.

A Systematic Survey of the Field of Games in Software Engineering Education

So far, games have been discussed in a rather generic fashion. A necessary precursor for this project was find out what games already existed in the field of software engineering education, how effective they had been, and how Simsoft

might be able to contribute new knowledge. To this end a systematic review of the literature was conducted using a collection of online science, engineering, education, and business databases looking for games or simulations used for educational or training purposes in software engineering or software project management across any of the Software Engineering Body of Knowledge (SWEBOK (Bourque et al., 1999) knowledge areas.

For this survey, an established procedure for conducting systematic reviews in the field of software engineering was followed (Kitchenham, 2004), and which has been used to survey the game field before (Gresse von Wangenheim & Shull, 2009). Given the upward trend in the use of games for software engineering education revealed in that previous survey, it was timely to update and expand the search.

Data Sources and Search Strategy

To perform this review the IEEE Xplore Digital Library, the ACM Digital Library, ScienceDirect, Sage Journals Online, ProQuest, the ISI Web of Knowledge, and the Wiley Online Library were used. The following pseudo-code search string was adapted for the specific query languages of each library:

```
where abstract OR title OR keywords contain (  
  ((game OR simulation) AND (learning OR teaching OR education OR training))  
  AND  
  (software engineering OR software project OR  
  software process OR software design OR  
  software testing OR software configuration management OR  
  software quality OR software management OR  
  software maintenance OR software construction OR  
  software requirements OR  
  software engineering tools and methods))  
AND  
(date >= 1990)
```

That is, we looked for games or simulations (computer and non-computer based) used for educational or training purposes in software engineering or software project management across any of the SWEBOK knowledge areas. (Despite the distinction made between game and simulation in the introduction, the terms are often used interchangeably in the literature (Maier & Grossler, 2000), therefore *simulation* was used as one of the search parameters).

Inclusion and Exclusion Criteria

The results were limited to English-language papers published from 1990 to the present in peer-reviewed journals and conference proceedings. (Only two significant studies pre-date 1990: (Boguslaw & Pelton, 1959; Horning & Wortman, 1977); these were excluded because of their distance in time and to allow a more direct comparison with the (Gresse von Wangenheim & Shull, 2009) survey). Excluded were position papers, papers in which no data was reported (unless they were preliminary papers for completed studies), and those in which the game or simulation was not used to train or educate the players at a tertiary level.

Study Identification and Selection

The initial database searches returned a total of 243 papers. The titles and abstracts were analysed according to the inclusion and exclusion criteria, and any off-topic papers were discarded. This left 36 papers, which were grouped according to the study they described.

Data Extraction

Each paper passing the selection process was read in depth and the following data was extracted:

- A brief description of the game and how it was played.
- The experimental design used by the study, which could be either true experimental (random assignment and comparison with a control group), quasi-experimental (comparison with a control group only), or non-experimental.
- The number and type of the players.
- The type of research tool used to collect the data, for example questionnaires, observation, pre- and post-test surveys.
- The primary SWEBOK knowledge area on which the game is focussed. The SWEBOK attempts to characterise and bound the software engineering body of knowledge; the ten knowledge areas are the major topical divisions within the field.
- The expected learning outcomes classified according to Bloom's (1956) cognitive domain taxonomy.
- The principal findings of the study.
- The country in which the game sessions were conducted.

Appendix L: Full Data Extract of Games Used in Software Engineering Education shows the full data extract of 36 papers describing 26 studies.

Survey Results

Appendix L shows that the preferred medium for games in the field is single-user computer-based (22 out of 26) rather than board and card games. Although computer games are easier to distribute and administer across a large number of players, some of whom may be in remote locations, the rich and sometimes complex user interfaces of these computer games often contributed little to the stated learning objectives. Figure 7 also shows that most of the studies were non-experimental (16 out of 26) meaning they didn't use control groups nor randomly assign participants to different groups.

The survey results showed that games have been used in a variety of ways to teach different aspects of software engineering and software project management. Appendix M shows the distribution of games across the world based on the SWEBOK knowledge area they were designed to address. Most games (21 out of 26) focused broadly on software engineering management or the development process and most activity (21 out of 26) occurred in Europe and the Americas.

Overwhelmingly, the learning objectives of the games found by this survey were pitched at the first rung of Bloom's taxonomy, knowledge. In general, those studies that assessed the degree of learning by the participants found that the participants

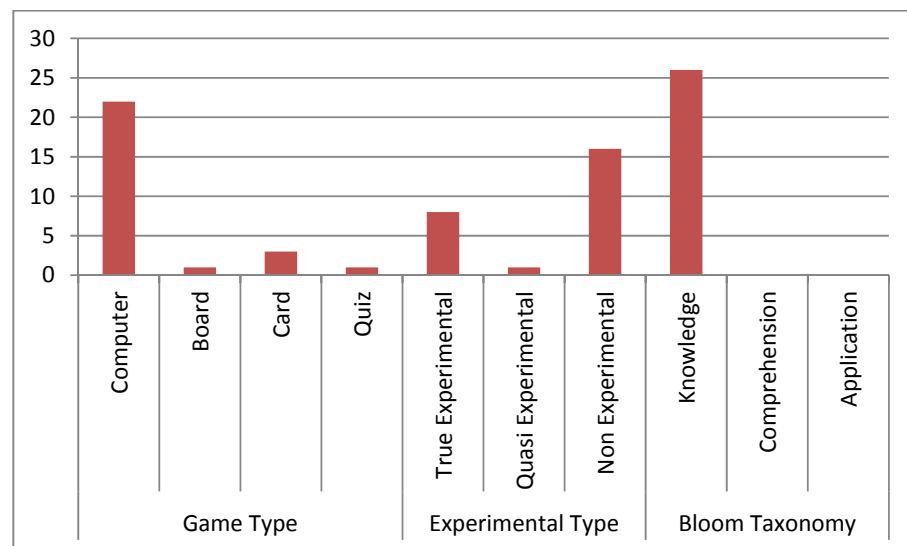


Figure 7: Game surveys classified by game type, experiment type, and Bloom taxonomy

were sometimes learning new concepts, but they were mainly reinforcing known theories. Only Navarro (2009) and Hainey et. al. (2010) evaluated the effectiveness of games for players of different skills and backgrounds and each found that games were suitable for a wide variety of participants.

It should be noted, however, that apart from Navarro's and Drappa and Ludewig's body of work, many of the research projects in Appendix L had very small sample sizes and few others were developed or repeated beyond that described in the initial papers. Both small and large research populations were made up exclusively of students.

The studies in Appendix L will be revisited in the Discussions chapter where a detailed comparison is made with the game developed for this project.

Summary

Software development is an inherently complex endeavour because of both the ephemeral qualities of software itself and the dynamic socio-technical system in which it is developed. These essential and accidental qualities of software are often the root cause of many quality and productivity issues. But, steps are being taken to make software development more reliable, more predictable, and more like its engineering namesake:

- A body of knowledge, the SWEBOK, has been defined which captures accepted practice in the field and which also forms the basis of curriculum development and accreditation, licensing and certification programs.
- Standards of ethics and conduct have been developed to guide software engineers in responsible behaviour, although these are still optional and unenforceable.
- Professionally-endorsed curriculum guidelines for graduate and post-graduate software engineering education have been developed to meet the latest technical developments and evolving industry demands.

Of interest for this research project is the way software engineers are educated because this directly and significantly affects so many other areas of professional practice. In the curriculum guidelines considered here, each identifies better software project management skills and better soft, or peopleware, skills as critical

for all graduating students, but the guidelines are intentionally vague on how institutions should teach these. It is in the final, synthesising capstone project that such skills are most needed, yet these projects are usually the final unit of study for a degree which leaves little time for reflection or remedial action.

One possible way to tackle these problems is to use a serious game— a game designed to teach and educate players about some of the dynamic complexities of the field in a safe and inexpensive environment. Importantly, games are not one-shot opportunities in the way capstone projects are: a game can be played, studied, tweaked, and replayed as many times as needed. Games have been used this way in many different business, military, and social environments, and have proven to be efficacious. They also draw their intellectual integrity from sound education theory such as problem-based learning meaning they have an authority to be taken seriously as learning and research tools.

This is not the first research project to look at games in software engineering education. A systematic survey of the field discovered over two dozen studies using mostly single-user computer games to teach various aspects of the software development lifecycle. However, few of these games were developed or repeated beyond their initial implementations, which suggests they weren't sufficiently able to demonstrate their pedagogical value to warrant further effort. For example, the survey revealed some common themes:

- Many of the games in the field were overly complex, with rich user interfaces that contributed little to the stated learning objectives.
- Single-user games were being used to demonstrate team practices.
- The research populations were made up exclusively of students, which made it difficult to extrapolate the use of games in other education and training situations.

An opportunity therefore exists to explore more fully if and how games— and games that address the above design themes— can contribute to better software engineering management and help fill some of the pedagogical gaps in the current curriculum guidelines.

Chapter 3— Research Methodology

Introduction

The purpose of this research project is to see if and how games can contribute to better software engineering management education by helping software engineers and project managers explore some of the dynamic complexities of the field in a safe and inexpensive environment. If games *can* contribute, then what features make them most efficacious? To this end, this research project addressed the following research questions and related hypotheses:

Can games contribute to better software engineering management education? (Q1)

Games built on sound software project management principles are a more effective means of improving software project management education than more traditional pedagogical means. (H1)

If games are to contribute to better software engineering management education, what features make them most efficacious? (Q2)

For best effect, players should be able to easily relate the context of a game to their real-world experience. (H2.1)

For best effect, games should be simple to play and understand and only as theoretically complex as needed to explore the concepts at hand. (H2.2)

This chapter describes the research methodology used here by looking at these areas: the rationale for choosing the research approach; a description of the research sample; a description of the data collection methods; how the data was analysed and synthesised; the ethical considerations of the project; and the limitations of the research. The chapter then concludes with a brief summary.

Rationale for a Qualitative Research Design

When making decisions about the design of a research project, a researcher generally makes a choice between quantitative and qualitative methods, or perhaps a mixture of the two. To first define the terms:

Quantitative research is a means for testing objective theories by examining the relationship among variables. These variables, in turn, can be measured, typically on instruments, so that numerical data can be analysed using statistical procedures. (Creswell, 2009, p. 4)

Meanwhile, qualitative research does things differently:

Qualitative research is a means for exploring and understanding the meaning individuals or groups ascribe to a social or human problem. The process of research involves emerging questions and procedures, data typically collected in the participant's setting, data analysis inductively building from particulars to general themes, and the researcher making interpretations of the data. (Creswell, 2009, p. 4)

The above quotes are heavy with meaning so it's worth seeing the two methods side-by-side as in Table 6. From this comparison, it seems the former is best used when we need to understand the factors influencing an outcome and we can design an objective experiment to test known variables; while the latter is best used when we are less sure of the key variables and we have to do some exploration along the way.

Because this research project is exploratory rather than deterministic and is seeking to understand a complex socio-technical system (software engineering management), a qualitative research approach was used. Within this qualitative paradigm, grounded theory was used because it is a means of inductively developing a theory from the collected data (Lincoln & Guba, 1984, pp. 204 – 208; Strauss, 1987; Maxwell, 2004, pp. 42 – 43). Sherlock Homes would have appreciated grounded theory: “It is a capital mistake to theorize before one has data. Insensibly one begins to twist facts to suit theories, instead of theories to suit facts” (Conan Doyle, 1974, p. 19).

The Researcher

Because the researcher in qualitative projects is a key player in collecting the data on which the findings are based, it is reasonable that any biases, assumptions, and personal values are made clear from the start. To this end, the following is a brief résumé of my education and professional experience.

In 1996 I completed a Bachelors degree in computer science and in 2002, a Masters degree in software engineering. My Masters research project (Caulfield,

2001) compared a range of system thinking methodologies before making a case that system dynamics was most suited to ill-defined problems that require some mixture of quantitative and qualitative analysis.

I also hold a number of technical certifications in Java (web development, business component development, and web services); XML; object-oriented analysis and design; MySQL (database developer and administrator); DB2 (database developer and administrator); RPG programming; and WebSphere application server administration.

Table 6: A comparison of quantitative and qualitative research designs.

Quantitative Research	Qualitative Research
"There's no such thing as qualitative data. Everything is either 1 or 0" (F. Kerlinger in Miles & Huberman, 1994, p. 40)	"All research ultimately has a qualitative grounding" (D. T. Campbell in Miles & Huberman, 1994, p. 40).
Philosophical Worldview	
Post-positivist, deterministic, reductionist, empirical observation and measurement of specific variables.	Constructivist, social and historical construction, variables are seen as complex, interrelated and difficult to measure.
Strategy of Inquiry	
<ul style="list-style-type: none"> • Experimental designs. • Non-experimental designs such as surveys. 	<ul style="list-style-type: none"> • Narrative research • Phenomenology • Ethnographies • Ground theory studies • Case studies
Approach	
<p>Begins with theories and hypotheses and uses instruments, experimentation, and deductive logic.</p> <p>The results are written up in an abstract, formal manner.</p>	<p>Theories and hypotheses emerge over time, uses inductive logic, and the search is for patterns.</p> <p>There are multiple sources of data such as observations, interviews, field notes, documents.</p> <p>The results are written up as rich descriptions.</p>
Research Methods	
<ul style="list-style-type: none"> • Analysis of performance, attitude, census, or observational data. • Statistical analysis and interpretation. • Pre-determined approaches 	<ul style="list-style-type: none"> • Participant observation. • Text, document, and image analysis. • Interviews using open-ended questions. • Emergent approaches.
Role of the Researcher	
Detached and objective	Personally involved and empathetic.

Sources: (Lincoln & Guba, 1984; Miles & Huberman, 1994; Maxwell, 2004; Guba & Lincoln, 2005; Creswell, 2009)

Since 1996 I have worked for a number of large companies in areas such as banking, agribusiness, defence, and state government; first as a business analyst, then as an analyst/programmer and senior Java software engineer. I am currently working as a lead development analyst for a consulting software company contracted to a Western Australian health insurance company. The work here involves small teams (two to four people) developing Java enterprise applications. Largely based on my work in these places, I have published a range of articles in industry magazines and on technical websites (see for example Caulfield, 2005, 2006a, 2006b, 2009b, 2009a).

I am a member of the IEEE Computer Society, the Association for Computing Machinery, and the Australian Computer Society.

My education and work experience have given me with a rich insight into software project management; naturally, they also mean I bring certain biases to the way this study has been designed and how I collected and interpreted the data. I commenced this study with a belief that software engineering management is generally poorly executed, with the root causes usually being managerial and political rather than technical. Even so, throughout the course of the study I have tried to engage in critical self-reflection by consciously evaluating opinions at dissonance with my own and by seeking the views of colleagues and advisors. Various procedural safeguards (discussed in the following section called Reliability, Validity, and Applicability of the Findings) have also been put in place to obviate my subjectivity.

The Research Sample

Purposive sampling (Lincoln & Guba, 1984, p. 40; Patton, 2002) was used to select the participants of the study from the following pools:

- Post-graduate project management students from Edith Cowan and Curtin universities.
- Software engineers, project managers, and account managers from a Perth-based software consulting company.

A call for participation was distributed by email and the participants replied if they wished to take part. Snowball sampling (Marshall, 1996, p. 523) was allowed, whereby those reading the email were encouraged to refer others in the same field they thought would be interested in taking part.

Although the participants each had an information technology or project management background, they exhibited notable variances in experience (from recent graduates to 25-year industry veterans); skills (from those still studying to highly-certified professionals); and cultural diversity (the participants came from Australia, Europe, the Middle East, Asia, and South Africa).

The participants (59 in total) joined one of seven game sessions held between May and September 2010.

Ethical Considerations

For any research project of this type, dealing with the participants in an ethical manner is a prime consideration (Oliver, 2003). The participants give freely of their time so that the body knowledge can hopefully be advanced, so it is incumbent on the researcher to tell them what the project is trying to achieve, and to respect any concerns they might have about how the resulting data will be used.

Although the nature of this research was considered benign and wasn't thought to pose any ethical threats to the participants, various strategies were used to inform the participants about the project, to protect their privacy and rights, and to address any concerns they might have:

- Potential participants were given an information letter that explained the research project, what they would be asked to do, and how long it would take (see Appendix F: Information Letter to Participants). If they wished to proceed, which was entirely voluntary, each person completed and signed a consent form (see Appendix G: Informed Consent Document). Even after the consent form was returned, participants were free to withdraw at any time.
- Individuals were not identified in any way in the research data. To help link specific game performance with the responses in the post-game questionnaires, Simsoft generated a random reference number when a

game session was started and the participants were asked to enter this when they completed the online survey.

- Precautions were taken to secure the research documents and data, and these were only available to the researcher and the project supervisors. These records will be destroyed when the requirements of this project are completed.

Participants were given the contact details of the researcher, the project supervisors, and the university's ethics officer in case they needed further information.

Data Collection Methods

Controlled experiments in software project management are rare and only one is known to have been attempted:

A controlled experiment... One project with lots of pressure and one with almost none, all three charged with doing the exact same task. We could watch them to see which one finished first... We could set up one group with a staff that was too big, another with a staff that was too small, and a third one that had just the right number... One staffed by people that have worked together before, pitted against another team staffed with strangers. (DeMarco, 1997, p. 25)

Unfortunately, DeMarco's controlled experiment is a work of fiction. A project manager, Webster Tompkins, is kidnapped by a benevolent dictator (a thinly-disguised Bill Gates) and is given virtually unlimited resources to create re-branded and reverse-engineered versions of six well-known software products. Three teams, of different makeup, independently attempt each product, making eighteen teams in all. The teams are then tracked for performance and quality.

While this conforms to the tenets of good experimental design, cost alone makes it unrealistic. Nevertheless, there is an alternative— use a game, where the consequences of failure are lower and where exploration and risk-taking are encouraged to see what benefits they might bring (Gee, 2007b, p. 216).

In this spirit, a game called Simsoft was developed for this research project. Games such as Simsoft have typically been deployed in university courses in one of three ways:

-
- Teach sections of the same course using different methods, and then compare the results of students on a common test. For example, all students in a course might attend a common lecture, and then attend either a traditional tutorial session or a tutorial that uses a game (see, for example McKenney, 1962; Raia, 1966). Besides the difficulty of obtaining adequate control of factors such as student composition, instructor characteristics, and grader evaluations, previous studies in this vein have shown that students participating in games are obviously learning ‘something’ other than what the traditional method might teach them, but that ‘something’ cannot be measured by a common test (Parasuraman, 1981, p. 192).
 - Evaluate the student’s grades or scores in the simulation part of the course with their grades in other more traditional assignments and examinations in the course (see, for example Remus, 1977; Remus & Jenner, 1981). There are some conceptual and methodological problems with this approach. For example, a student’s game score might reflect their ability to play or beat the game rather than their decision-making ability (Parasuraman, 1981, p. 194).
 - Sample players subjective attitudes regarding the usefulness of the games before, during, or after play, or a combination thereof, by obtaining written feedback (see, for example Jackson, 1959; Dill & Doppelt, 1963; McKenney & Dill, 1966; McKenna, 1991; Herz & Merz, 1998). This technique can be criticised on the basis of “how qualified are college students, with little or no practical business experience, to make any meaningful evaluation of business simulation games?” (Parasuraman, 1981, p. 194). Other studies have shown that student performance in games, when compared to that of experienced managers, raises serious questions about how much can be generalised to behaviour patterns in the business world (Babb et al., 1966, p. 471).

Each approach therefore has its strengths and weaknesses and is part of a broader debate concerning the value of games as pedagogical devices and when compared to other methods of instruction (Amstutz, 1963; Moore, 1967; Boocock, 1970; Moskowitz, 1973; Hand & Sims, 1975; Wolfe & Guth, 1975; Parasuraman, 1981; Remus & Jenner, 1981; Prohaska & Frank, 1990; Gredler, 1996, 2004).

To address some of these issues, an approach similar to the third option above will be followed. Simsoft game sessions were conducted for teams of post-graduate project management students (for software and general projects), and practising

software project managers and developers. Overall game evaluation was made on the basis of performance in Simsoft, pre- and post-game surveys, and a qualitative rich analysis of the interactions that were observed during the game sessions.

Simsoft Overview

Physically, Simsoft comes in two pieces:

- An A0-sized printed game board (see Appendix C: Simsoft Game Board) around which the players gather to discuss the current state of the project and to consider their next move. The board shows the flow of the game while plastic counters are used to represent the staff of the simulated project. Poker chips represent the team's budget, with which they can purchase more staff, and from which certain game events may draw or reimburse amounts depending on decisions made during the course of the game.
- A simple Java-based dashboard (see for example, Sterman, 1988; Langley et al., 1999; Caulfield et al., 2011b), through which the players can:
 - See the current and historical state of the project through a series of simple reports, messages, and other information.
 - View the underlying system dynamics model so they can be fully informed about the relationships behind particular game variables (Machuca, 2000).
 - Can adjust the project's settings, for example to recruit new staff, before advancing the game's time to create the state of the project.

The aim of the game is to complete the project on time and with funds (poker chips) left over.

The engine behind Simsoft is a system dynamics model which captures a small set of fundamental causal relationships in software engineering projects. System dynamics has been used for this task before and a case has been presented in other work (Caulfield, 2001; Caulfield & Maj, 2001, 2002, 2007).

Behind the system dynamics model will be a relational database (see Appendix H: Simsoft Database Design) to store the decisions entered by the players, and which will capture the state of the model at each time slice. This will allow the game to be rolled backward or forwards, replayed, and studied.

Simsoft was designed in this way primarily in response to some of the perceived shortcomings of extant games in the field. For example, most of the other games are overly complex, with rich user interfaces that contribute little to the stated learning objectives. Therefore, Simsoft uses a large game board and simple calculator. And, most others are single-user games trying to demonstrate team practices, whereas Simsoft is predicated on team play.

Game Sessions

The players are formed into teams of two or three or more and they decide on a name for their team. Each team is given a scenario (see Appendix D: Simsoft Instructions to Players) that describes the requirements for a small software development project. Taking the role of project manager, the team must manage the project from start-up to final delivery.

What the players must deliver is handled by boxes on the left side of the Simsoft game board (Figure 8).

At the start of the game there is a pool of work to do. This pool is represented on the game board with small plastic counters in the *Work To Do* box. These counters can be thought of as Use Cases or items in a work breakdown structure; whatever is most familiar to the players. Depending on the resources available to do the work, the units of work (the counters) move from the *Work To Do* box to a *For Review* box, where the work is reviewed before passing to the *Completed Work* box. Not unexpectedly, some work will fail the review and go to the *Rework* box, before passing back to *For Review* and trying again to get to *Completed Work*. The team can reduce the amount of rework by 'buying' more quality assurance staff (staffing is considered shortly).

The work-to-do, review, rework, work-completed cycle is a fundamental project work structure first discussed and modelled by Roberts (1964). Roberts' initial work has been expanded greatly by subsequent researchers who have added rich details based on actual projects (see Lyneis & Ford, 2007 for a comprehensive survey of the field), but the underlying work structure remains unchanged.

Based on the starting scenario of the game, information provided during the game, and their own real-world experience, the players make decisions about how to proceed: whether to hire more staff, what hours should be worked and so on. The team is given a budget for the project (poker chips), with which they ‘buy’ more staff. But, there are trade-offs: more experienced (and therefore more productive) staff are more expensive (New Hires are \$500, Quality Assurance are \$600, Mid-Rangers are \$700, and Old Hands are \$1000), and the staff do not become available immediately— there are recruiting delays to be considered (Yourdon, 1998, p. 98).

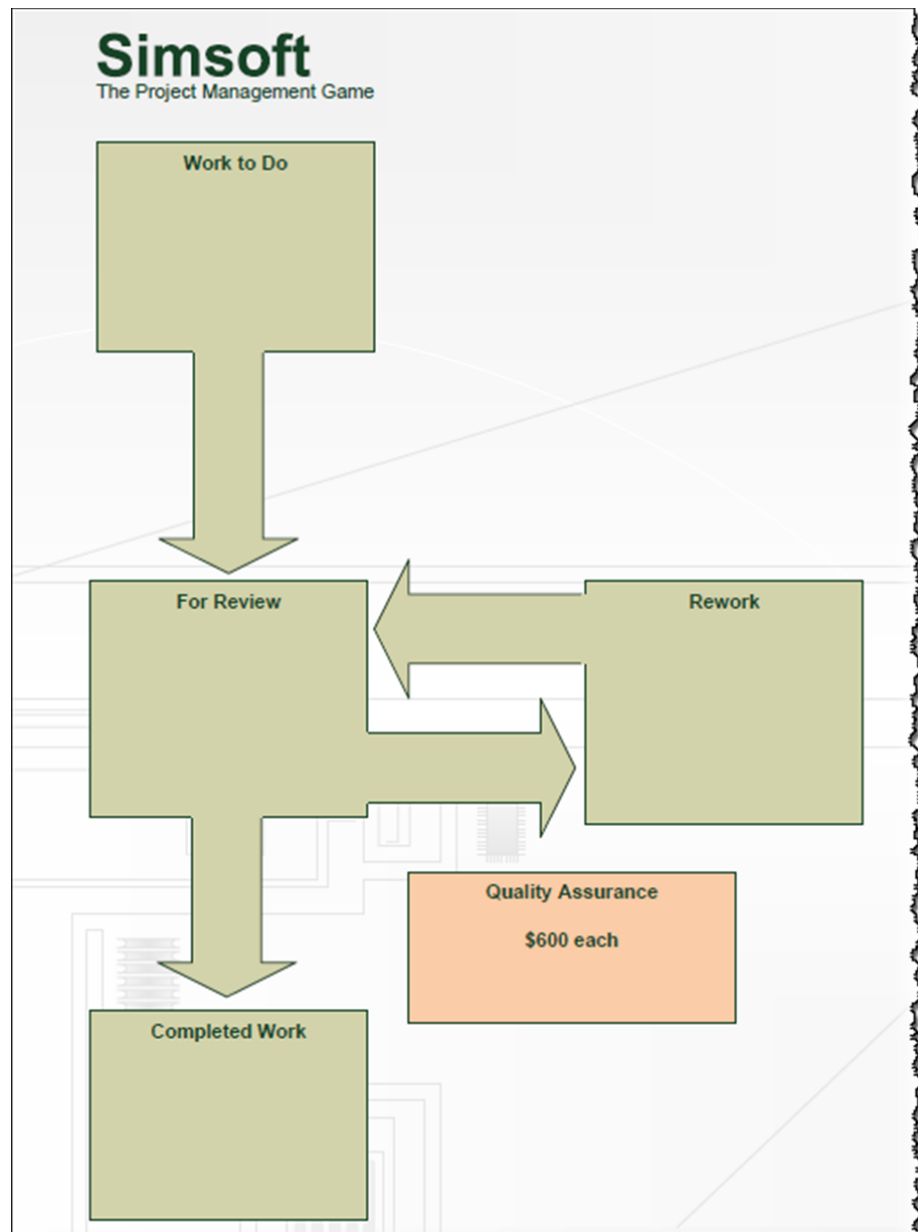


Figure 8: Units of work boxes on the left-hand side of the Simsoft game board.

The players can also see from the game board (Figure 9), that staff naturally gain experience (and therefore become more productive) as the project proceeds—

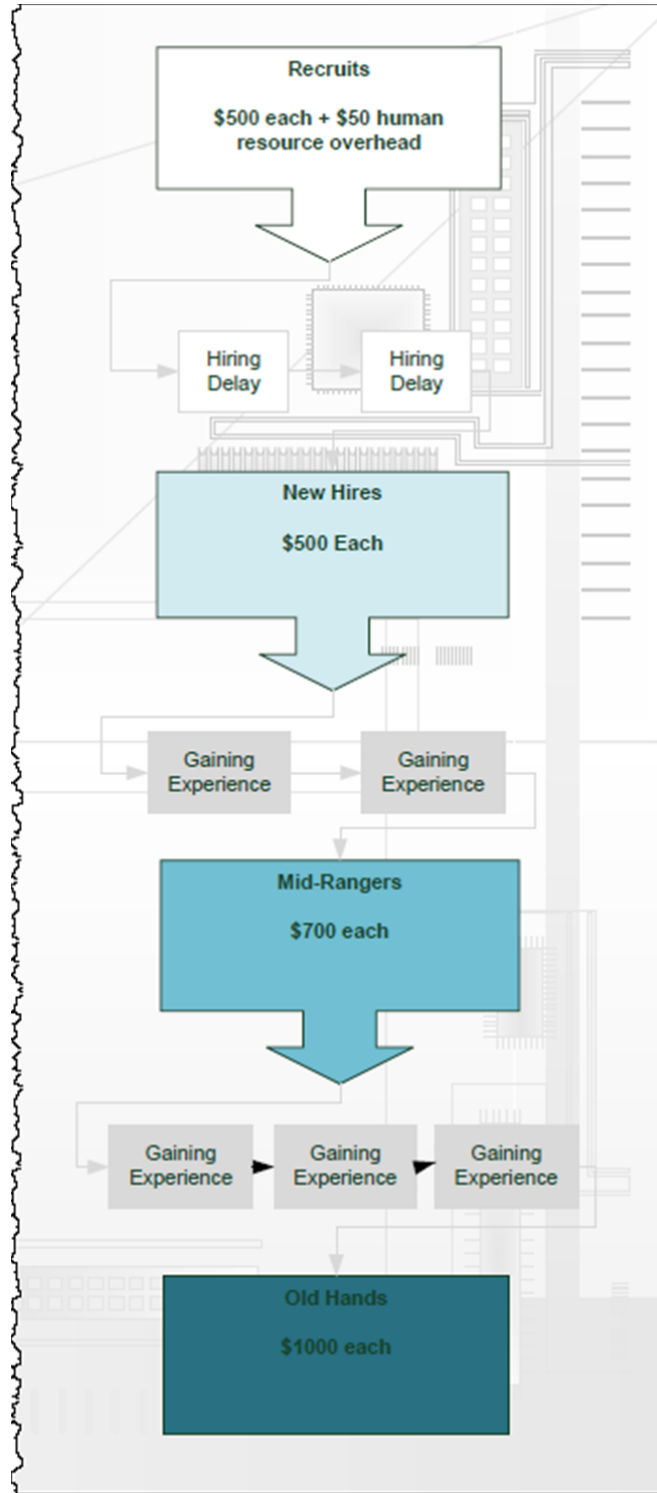


Figure 9: Resource boxes on the right-hand side of the Simsoft game board

something further they need to consider before spending their precious budget chips.

These decisions are entered through the software dashboard, project time is advanced by one week, and the dashboard tells the participants which pieces to move around the board (Figure 10).

The screenshot shows the Simsoft dashboard interface, titled "Simssoft Calc". It features a menu bar with "File", "Game", and "Help". The main content is divided into three sections:

- 1. Current Game State - Week Project Start**
 - Units of Work:** Four input fields for "Work to Do", "For Review", "Rework", and "Completed Work", each with a value of 0.
 - Human Resources:** Six input fields for "Recruits", "New Hires", "Mid Rangers", "Old Hands", "Quality Assurance", and "Working Week". "Recruits", "New Hires", "Mid Rangers", and "Old Hands" are set to 0. "Quality Assurance" is set to 0. "Working Week" is set to 0,00.
 - Project Report:** A large empty text area.
- 2. Enter Your Decisions**
 - Input fields for "Recruits", "New Hire Recruits", "Recruit Mid Rangers", "Recruit Old Hands", "Quality Assurance", and "Work Hours per Week...". "Recruits", "New Hire Recruits", "Recruit Old Hands", and "Quality Assurance" are set to 0. "Recruit Mid Rangers" is set to 0. "Work Hours per Week..." is set to 40.
 - A "Submit" button is located at the bottom right of this section.
- 3. Move These Pieces on the Board**
 - Units of Work:** A table with "From" and "To" columns and input fields for the number of pieces to move.

From	To	Value
Pool	Work to Do	0
Work to Do	For Review	0
For Review	Rework	0
Rework	For Review	0
For Review	Completed Work	0
 - Human Resources:** A table with "From" and "To" columns and input fields for the number of pieces to move.

From	To	Value
Recruits	New Hires	0
New Hires	Pool	0
New Hires	Mid Rangers	0
Recruits	Mid Rangers	0
Mid Rangers	Pool	0
Mid Rangers	Old Hands	0
Recruits	Old Hands	0
Old Hands	Pool	0
Recruits	Quality Assurance	0
 - An "Advance" button is located at the bottom right of this section.

Figure 10: Simsoft dashboard.

The game is now in a new state, which the participants must interpret and then decide their next move.

As in the real world, not everything runs smoothly in the Simsoft world and the players may need to rethink their plan. At random times, Simsoft will generate one of the following events:

- *A major design flaw has been discovered. Add 5 more units of work to the Rework box.*
- *Your team wins lotto and three staff have resigned, effective immediately. Remove three staff from the game board. (In this case, the counters representing the staff are removed from the board and put back into the pool).*
- *The Finance department have made a mistake. Collect \$500 from the bank.*

Events like these are called games pulses: an event outside of normal play that the teams must take account of when formulating their next decision set (Duke, 1980, p. 368; Schumann et al., 1996; Wolfe & Fritzsche, 1998). How the players react to these pulses will be revealed in their subsequent decision sets.

Play continues in this manner until there is no more work to do (all the unit-of-work counters are in the *Completed Work* box of the game board), or until the project deadline passes, whichever comes first. The aim of the game is to deliver the software before the deadline and on budget (with poker chips left over).

Each team will operate their project independently of other teams. There won't be any competition between the teams, but they will be able to see overall performance of the other teams. While it will be possible to compare team performance in the game according to indicators such as budget and timeliness, any such ranking will be nominal since the degree to which the participant's meet the game's states learning objectives will be the key performance determinant (Greenlaw & Wyman, 1973).

Game Administration

Simsoft game sessions will be overseen by an administrator whose role will be to:

-
- Explain the learning objectives to the participants. The players must be made familiar with the decision-making environment created by the game, the type of decisions that will be required, and the quantifiable indicators of effective decision making (Watson & Blackstone, 1989, pp. 494 - 496).
 - Ensure that team composition is fair and not left to self-formation. McKenney and Dill (1966) recommended that homogeneous teams be avoided and that high-ability players be distributed throughout the groups. The ideal team size is three since four-member teams experienced more interpersonal problems, while anything larger could actually jeopardise the completion of the game itself (Strother et al. cited in Greenlaw & Wyman, 1973, p. 274).
 - Provide the teams with feedback and technical assistance during the decision rounds.
 - Run the after-game debriefing session that helps relate elements of game play with the learning objectives.

For each of the game sessions, the administrator was the primary researcher.

Pre- and Post-Game Surveys

Before the game sessions, the players completed an online survey (see Appendix J: Post-Game Survey) designed to test their knowledge of general software engineering and project management principles. The survey questions were drawn from examination preparation guides for the IEEE's Certified Software Development professional certification (Naveda & Seidman, 2006) and the Project Management Professional certification (Heldman, 2007).

After the game sessions, the players were asked to complete another online survey. Post-game surveys are a common feature of game research (Faria, 1987; McKenna, 1991; Eldredge & Watson, 1996; Faria, 1998; Faria & Wellington, 2004) and in problem-based learning (Tang et al., 1997), the key foundation of Simsoft's design.

Based on these exemplars, a survey was designed (see Appendix J: Post-Game Survey) to gather basic profile data (industry experience, any previous experience playing serious games), the players' perceptions of the value of the game helping to understand some of the dynamic complexities of software project management, and to assess what they might have learned during the game.

Data Analysis and Interpretation

The analysis and interpretation of the data for this project followed a path used many times before in qualitative research: collect the data, analyse it for themes or perspectives, and then report on four or five of those themes (Lincoln & Guba, 1984, p. 339; Bloomberg & Volpe, 2008, p. 100; Creswell, 2009, p. 184). These steps are expanded in Figure 11 (Creswell, 2009, p. 185).

In more detail, the following steps were taken:

- Organised and prepared the data for analysis by transcribing the interviews,

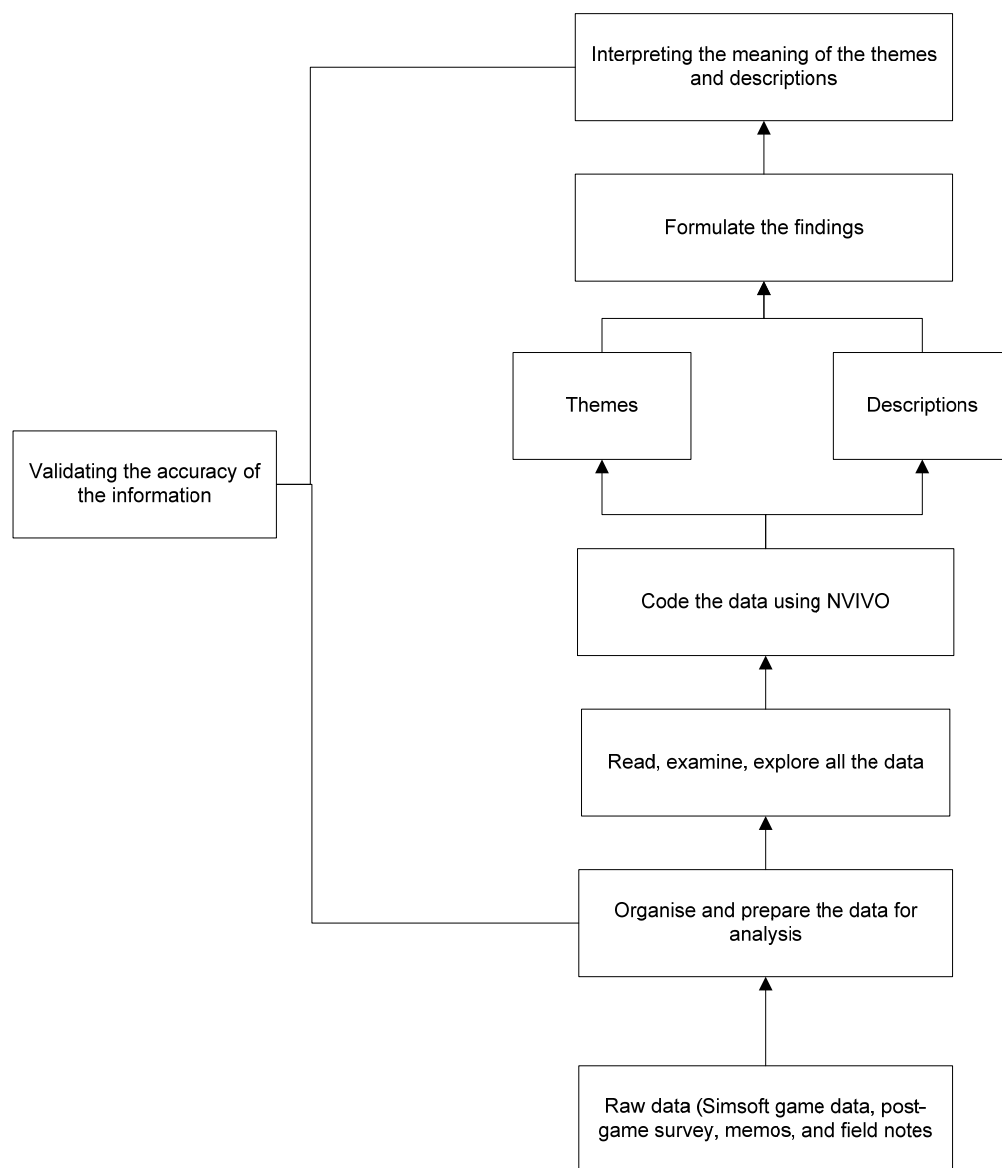


Figure 11: The process of analysing and interpreting the data.

and writing up the field notes and memos (Creswell, 2009, p. 185).

- Re-read, examined, and explored the data to get a high-level sense of what had been collected.
- Started to analyse the data by first coding it— breaking it into named chunks or categories that can then be used to make comparisons between things in the same category and to help develop theoretical concepts (Strauss, 1987, p. 29; Rossman & Rallis, 1998, p. 171). The software package NVivo (QSR International, 2010) , an industry-standard tool for collecting and analysing qualitative research materials, was used for this task (Richards, 2009, chapter 5).
- Used the coding to identify and describe themes and patterns in the data, which then became candidates for more detailed analysis and, potentially, major findings of the project (Maxwell, 2004, pp. 96 – 99).
- Formulated the finding statements and supported these with specific data instances and then summarised the key findings.
- Sought meaning in the findings by linking it to experience, insight, or the literature. The most commonly asked question was: “What were the lessons learned?” (Lincoln & Guba, 1984).

The above step-by-step list and Figure 11 might give the impression that the analysis and interpretation of the data proceeded in a linear fashion once all the data had been collected. In reality, the process was highly iterative and started as soon as the first data was available— a feature common to this type of research (Lincoln & Guba, 1984, pp. 241 – 242).

Reliability, Validity, and Applicability of the Findings

Compared to objective, deductive quantitative research, qualitative research has often been called undisciplined, sloppy, merely subjective, and indiscriminately responsive to the loudest bangs and brightest lights (Lincoln & Guba, 1984, p. 289). Add to this a researcher intimately involved in the data collection and carrying certain biases, and it is natural to question the trustworthiness of the results:

The basic issue in relation to trustworthiness is simple: How can an inquirer persuade his or her audiences (including self) that the findings of an inquiry are worth paying attention to, worth taking account of? What arguments can be mounted, what criteria invoked, what questions asked, that would be persuasive on this issue? (Lincoln & Guba, 1984, p. 290)

That is, how can we demonstrate the findings are reliable, valid, and applicable?

Reliability

Reliability is “the extent to which a measurement procedure yields the same answer however and whenever it is carried out” (Kirk & Miller, 1986, p. 19), or if someone other than the current researcher carried the same process, would they come to the same conclusions?

For this project, the coding scheme was peer reviewed by an independent party— a skilled qualitative researcher from a local government department who was also an accredited NVivo trainer.

Validity

Validity is the “degree to which the finding is interpreted in a correct way” (Kirk & Miller, 1986, p. 20), which is also known as internal validity or, in Lincoln and Guba’s (1984, p. 290) classic definition, “truth value”.

Internal validity for this project was addressed in a number of ways:

- More than one data source was used so that the results could be triangulated.
- The researcher’s education, work experience, and biases have already been made known (see the previous section The Researcher).
- Member checking: the findings were reviewed by a random sample of four players from the games sessions.

Applicability

Applicability— or transferability, generalisability, external validity— is “the extent to which the findings of a particular enquiry have applicability in other contexts or with other subjects” (Lincoln & Guba, 1984, p. 290)? Some make the argument that this is really a burden for subsequent researchers who try to apply the findings in a new context, something known as the second decision span:

The first decision span allows the researcher to generalize the findings about a particular sample to the population from which that sample was drawn... The second decision span occurs when another researcher wants

to apply the findings about a population of interest to a second population believed or presumed to be similar enough to the first to warrant that application. (Marshall & Rossman, 2005, p. 202)

In light of the opposing ideas in this area, the best way to address the applicability of the findings— and the one used here— is to provide rich, detailed descriptions that will allow subsequent researchers to determine if the findings are relevant to their particular setting (Merriam, 1998).

Limitations of the Study

This research project has a number of limitations. Some of these are related to this project's own design, while others are common to all types of qualitative research. For example, perhaps the foremost limitation of any qualitative research project is the intimate involvement of the researcher in collecting and analysing the data: bias and subjectivity are reasonable charges. Aspects of the qualitative method that may limit the research (and their mitigations) were covered in the previous section; for those related to the project design, care has been taken to ensure these limitations are justified.

Simsoft is only focussed on human resource aspects of software engineering management, rather than, say, technical or methodological aspects. Apart from verisimilitude when compared to a real software project, this may not be a limitation because human resources— peopleware as DeMarco and Lister (1999) call it— is the one area that has most potential to offer an order of magnitude productivity improvement— a silver bullet. Consider the cover of Boehm's often-cited *Software Engineering Economics* (Boehm, 1981): it shows a bar graph of those factors that influence software project productivity such as programming practices, reliability requirements, and product complexity. By far the most influential factor is personnel/team capability. According to Boehm's many studies, understanding and improving this aspect of software projects offers the richest productivity rewards. It makes sense, then, for Simsoft to concentrate on peopleware at least in this first iteration of what will be ongoing research.

It can also be said that the Simsoft simulacrum is too simple because the players don't have to do any requirements analysis, software design or estimation, or any of those other things that are part of a full-breadth development cycle. But, there are advantages in keeping things simple. While deep and complex games create a

rich learning environment, players can be intimidated by this complexity and may be unprepared or unwilling to devote the necessary effort (Wolfe, 1978, p. 152). Other researchers (Raia, 1966, p. 351; Watt, 1977, p. 2; Butler et al., 1979; Meadows, 1999; Dempsey et al., 2002) support Wolfe's findings that simple games can provide essentially the same benefits as more complex games while avoiding player mortality (boredom and dropout).

Summary

This chapter has provided a detailed description of this project's research methodology. A qualitative research approach based on grounded theory was chosen because it provided the best way to tackle a research area without clearly defined experimental variables.

The research sample was made up of 59 purposively selected project management and software engineering practitioners and students—the players.

A simple game of a software project, Simsoft, was used as the primary research tool. Simsoft's design was based on perceived shortcomings in extant games in the field, such as their overly rich and complex user interface and their single-player nature. Before and after the game sessions, players completed surveys designed to see what they may have learned and how they experienced the game. Therefore, this research drew from multiple data sources: the Simsoft game database (see Appendix H: Simsoft Database Design for the data items captured), the pre- and post-game surveys (see Appendix I: Pre-Game Survey and Appendix J: Post-Game Survey), interviews with the players, researcher memos (Maxwell, 2004, p. 12), and field notes.

The research data was analysed using conventional qualitative methods, comparing it against the literature of the field and searching for emergent themes. The reliability, validity, and applicability of the data were addressed by various methods such as peer review and rich descriptions of the context and proceedings of the study.

Chapter 4— Findings

Introduction

The purpose of this research project was to see if and how games can contribute to better software engineering management education by helping software engineers and project managers explore some of the dynamic complexities of the field in a safe and inexpensive environment. If games *could* contribute, then what features made them most efficacious?

This chapter presents the findings from a series of Simsoft game sessions conducted with teams of post-graduate project management students (for software and general projects), and practising software project managers and developers. The data sources for the findings were the participants' performance in Simsoft, pre- and post-game surveys, interviews with the participants, and a qualitative rich analysis of the interactions that were observed during the game sessions.

Six major findings emerged from the research:

- There was evidence the participants were learning by doing.
- Games such as Simsoft are not sufficient learning vehicles by themselves and need to be supplemented by other methods.
- Simsoft is a suitable pedagogical device for participants of different skills and backgrounds.
- The majority of participants said they would be prepared to invest greater time and effort in games such as Simsoft if the reward was deeper understanding of a problem domain.
- The majority of the participants found that working in groups was a positive experience.
- The majority of participants preferred playing a board game to a fully computerised game.

The following is a thick description of these findings. Here, *thick description* is a loaded term and:

... does more than record what a person is doing. It goes beyond mere fact and surface appearances. It presents detail, context, emotion, and the webs of social relationships that join persons to one another. Thick description evokes emotionality and self-feelings. It inserts history into experience. It

establishes the significance of an experience, or the sequence of events, for the person or persons in question. In thick description, the voices, feelings, actions, and meanings of interacting individuals are heard. (Denzin, 1989, p. 83)

Verbatim quotations from the post-game survey and participant interviews are used to illustrate the findings where possible.

Because the data for this project came from multiple sources and was largely unstructured, NVivo (QSR International, 2010) was used to store all research materials and to then apply codes that revealed themes and descriptions and eventually the findings. SPSS (IBM, 2011) was used to analyse the survey and game data using a series of non-parametric statistical tests.

Descriptive statistics (for example, mean and standard deviation) were used to analyse the game features (for example, ease of navigation, help provided, and realism of the scenario) and raw game data (for example, games completed and abandoned, time taken). Mann-Whitney U tests compared knowledge levels in the pre- and post-game tests between the game teams and between player types (project managers, software developers, and students). Wilcoxon matched-pairs signed, ranks tests were used to compare the pre- and post game tests for each team and player type.

Finding 1— There was evidence the participants were learning by doing.

A key tenet of problem-based based learning, one of the theoretical foundations of Simsoft, is that when people work through problems for themselves, the knowledge they build *sticks* and they are more able to apply what they have learned in new situations. The following comments indicate that playing the game indeed helped the participants figure things out for themselves:

“Aha!”

“Our team figured out we could move more counters [work units] by investing in a couple of expensive, experienced developers, more middies, and some quality control people. Makes sense really”

“We spent our poker chips on lots of cheap newbies and before long had most of our counters [work units] in rework. We should have bought some old timers for guidance”

“Now I see why”

“I hadn't appreciated the level of productivity variability between developers before”

In addition to playing the game, all participants completed pre- and post-game surveys that included a number of questions designed to test their general level of knowledge about project management and software engineering concepts. Table 7 shows the results of the tests broken by the participants' role and years of experience. Each group performed better after playing the game. Two non-parametric statistical tests were run over the pre- and post-game results to determine if this improved performance was significant.

A Mann-Whitney *U* test ($Z = -1.091, p = 0.275 > 0.05$) indicated that there was no significant differences between the pre- and post-game results when considering

Table 7: Comparison of players pre- and post-game test scores

Role and Experience (in years)	n	Average Pre-Test Score (out of 8)	Average Post-Test Score (out of 8)	Difference Between Pre- and Post-Game Scores
Students	17	4.64 (SD = 0.861)	5.41 (SD = 1.460)	+0.77
0 to 1 years	17	4.64 (SD = 0.861)	5.41 (SD = 1.460)	+0.77
Software Developers	30	5.53 (SD = 0.995)	6.33 (SD = 1.107)	+0.80
0 to 1	0			
2 to 5 years	14	5.57 (SD = 1.089)	6.07 (SD = 1.268)	+0.50
5 to 10 years	11	5.72 (SD = 1.009)	6.818 (SD = .0750)	+1.098
10 to 15 years	5	5.00 (SD = 0.707)	6.00 (SD = 1.224)	+1.00
More than 15 years	0			
Project Managers	12	4.66 (SD = 1.497)	5.42 (SD = 2.020)	+0.76
0 to 1	0			
2 to 5 years	6	4.5 (SD = 2.073)	5.00 (SD = 2.529)	+0.50
5 to 10 years	1	5.00 (SD = NA)	6.00 (SD = NA)	+1.00
10 to 15 years	4	4.75 (SD = 0.957)	5.75 (SD = 1.892)	+1.00
More than 15 years	1	5.00 (SD = NA)	6.00(SD = NA)	+1.00
	59	5.10 (SD = 1.155)	5.88 (SD = 1.486)	+0.78

the broad groups of project managers, software developers, and students. A Wilcoxon signed ranks test ($Z = -1.604$, $p = 0.109 > 0.05$) also showed there was no significant difference between the pre- and post-game results of the three groups.

The same statistical tests were then run at a finer level of detail: against the years-of-experience sub-groups within the three main groupings of project managers, software developers, and students. Both the Mann-Whitney U test ($Z = -2.951$, $p = 0.003 < 0.05$) and the Wilcoxon signed ranks test ($Z = -2.552$, $p = 0.011 < 0.05$) showed there was a significant improvement between the pre- and post-game tests.

Together, these results indicate that while playing the game helped, none of the three main groups performed significantly better than the others. However, the years of experience a person has may affect how much they take from the game.

Finding 2— Games such as Simsoft are not sufficient learning vehicles by themselves and need to be supplemented by other methods.

While most players (40 out of 59) said that Simsoft helped put project management and software engineering theories into a practical context, the mean score was 2.64 out of 5 ($SD = 0.760$) when they were asked if games were a better way of learning and understanding technical material than through more conventional methods such as books, lectures, and case studies.

From an experienced software developer:

“I saw in the game aspects of theory covered at uni, but without knowing the theory first I probably wouldn't have recognised the significance.”

And these comments from two students:

“I was out of my depth”

“I could see the logic behind my team's decision, but I wouldn't have known enough to make the decision by myself.”

One project manager expressed an interest in using Simsoft as part of an undergraduate computer science course he teaches part-time, but:

“It would have to be used on the final weeks of the course when the students have some theory under their belt... Plus, there is little momentum behind problem-based learning at [my university] so the resources aren’t available to design a proper PBL based curriculum”

Table 7 also shows that the greatest improvement between the pre- and post-game tests was in those groups with the greatest work experience, so that relatively inexperienced participants took less from the game. This suggests that some level of *a priori* knowledge is needed for games like Simsoft to be truly effective.

However, when asked if games were a better way of more *thoroughly* learning a topic than through more conventional methods such as books, lectures, case studies, a significant minority (21 out of 59 participants) agreed or strongly agreed (mean = 3.00 out of 5, SD = 0.964). Self discovery seems to be the motive:

“I like to figure things out for myself”

On six occasions over the seven game sessions, the researcher overheard players saying they wished they could set Simsoft to match their work environment so they could game through some current issues.

Finding 3— Simsoft is a suitable pedagogical device for participants of different skills and backgrounds.

When asked if Simsoft was easy or hard to play (1 = too easy, 3 = about right, and 5 too hard), the majority of the participants (47 out of 59) felt that the game was a pitched at about the right level of difficulty (see Table 8).

This comment was from a student:

“Even though I’m still studying and don’t have much [practical work] experience, I was able to understand the game’s project and contribute to the decisions”

Table 8: Participants responses when asked whether they thought Simsoft was easy or difficult to play

Role and Experience (in years)	n	Average Response
Students	17	3.17 (0.528)
0 to 1 years	17	3.17 (0.528)
Software Developers	30	2.93 (SD = 0.253)
0 to 1	0	
2 to 5 years	14	2.92 (SD = 0.267)
5 to 10 years	11	3.00 (SD = 0.000)
10 to 15 years	5	2.80 (SD = 0.447)
More than 15 years	0	
Project Managers	12	2.58 (SD = 0.514)
0 to 1	0	
2 to 5 years	6	2.83 (0.408)
5 to 10 years	1	3.00 (SD = NA)
10 to 15 years	4	2.25 (SD = 0.500)
More than 15 years	1	2.00 (SD = NA)
	59	2.93 (SD = 0.449)

And, from a project manager with 10 to 15 years experience:

“[The] game was not too easy so that it was boring, but not too hard that newbies couldn't undetstad (sic) it.”

Across the seven game sessions there were no teams composed entirely of one group only, so each had a mixture of skills and experience. This was viewed positively:

“Our team had a mixture of abilities and life experience. I think this helped us make good choices”

“[One of our team] had read about Brooks’ model and could let us know if we were on the right track”

Finding 4— The majority (49 out of 59) of participants said they would be prepared to invest greater time and effort in games such as Simsoft if the reward was deeper understanding of a problem domain.

Many players said they reached the end of the game before they had time to fully explore the dynamics of the scenario, or they wanted to take more time discussing their options before committing to a decision. For example:

"The game was too short to discover what I wanted to know"

"I wanted to know more"

"We wanted more time to talk about our options"

The database of Simsoft game transactions showed that games lasted an average of 35 minutes ($SD = 7.082$) and that 80% of games finished within 40 minutes. The players were encouraged to stay behind after the game sessions to discuss and compare their results with other teams. Often, these after-game sessions lasted longer than the games themselves.

Considering the amount of time they had spent playing Simsoft, a majority of the players (49 out of 59) said they would be prepared to invest greater time and effort in games like Simsoft if the reward was greater understanding of the problem domain:

"What about running the game in real time, like the stock market game. That would give us time to make really considered judgements, people could be assigned research topics during the week"

"I hope that future versions will let me set up specific scenario and play them out. That would really help me in my work"

Outside of this research project, 10 players had previously participated in a long-running online stock market game in which notional shares were bought and sold based on actual prices published in a daily newspaper. Buy and sell decisions were submitted weekly and the team with the largest portfolio after three months was declared the winner.

Finding 5— The majority (44 out of 59) of the participants found that working in groups was a positive experience

An important component of many of the pedagogical theories behind Simsoft is team work, so it was important to assess how this was viewed by the players. A majority of players (44 out of 59) said they found it useful or very useful to work as a team and that this reflected how things often happened in the workplace:

“It was like [the agile] stand up meeting we have every morning”

“We organised our selves into roles we felt comfortable with or that fitted our day-job: someone on the calculator, someone moving the developer pieces, someone moving the units of work”

However, one student found something new in the practice:

“I thought software development was a solitary experience but it’s not really”

Others liked the opportunity to share opinions and learn from more experienced peers:

“Everyone had a chance to offer an opinion”

“I have little real-world project experience so it was good to get the advice of others and see how they approached problems”

But, as in any group activity, the game facilitator needs to be aware of cultural differences that may make some less inclined to contribute and of players who are dominating their groups:

“Generally, everyone had their say in final decision but a couple of times we were overridden”

Finding 6— The majority (44 out of 59) of participants preferred playing a board game rather than a fully computerised game

The players' responses to different features of the game were generally positive (Table 9).

Table 9: Players' evaluation of game features

Feature	Average (1 = very bad, 5 = very good; or 1 = strongly disagree, 5 = strongly agree)
Written instructions	Average = 4.44, SD = 0.771
The game was interesting	Average = 4.37, SD = 0.963
Realistic scenario	Average = 4.37, SD = 0.692
Navigation around the game	Average = 4.22, SD = 0.744
Game logic was apparent	Average = 4.18, SD = 0.730
Useful to work in teams	Average = 4.15, SD = 0.714
Prefer game-board version	Average = 3.98, SD = 0.754

Notable in Table 9 is that a majority of players (44 out of 59) preferred playing with a game board rather than a fully computerised version. Some typical comments were:

“The board game [was] simple and I could easily see the state of the game”

“When a group plays the game on a PC, someone controls the mouse and keyboard and they tend to dominate”

“Compared to computer-based games, the design was simple and we started playing without too much wasted time”

“Sometimes technology gets in the way”

“Everyone plays board games so we all knew what to do”

Outside of this research project, seven players had played The Beer Game, four-point distribution chain, originally developed at MIT and now used widely as a management educational tool in a variety of academic and commercial settings (Serman, 1989; Goodwin & Franklin, 1994; Senge et al., 1994; Lomi et al., 1997; Caulfield et al., 2004). In The Beer Game all calculations are performed by hand on simple worksheets. This found favour:

“Doing the calculations by hand means we have to understand”

“The calculator half of the game hides details. Just give us a calculator and we can work it out”

Although the players’ reception of the game was generally positive, clear written instructions were essential to make sure best use was made of the game session time. This comment was made by a player in the very first game session:

“Wasn’t sure of what we were supposed to do”

Initially, instructions for playing the game were handed out by the researcher after the players had completed the pre-game survey and just before they started the game. For the second game session onwards, a one-page instruction sheet was emailed to each player a couple of days beforehand so they could be prepared.

The database of Simsoft game transactions showed that only three games had to be abandoned and restarted. It was observed that once teams had made the first couple of decisions, they were able to continue without too much trouble.

Summary

This chapter presented the six findings of this research project which were discovered through a series of Simsoft game sessions conducted with teams of post-graduate project management students, and practising software project managers and developers. Data from the participants’ performance in Simsoft, pre- and post-game surveys, interviews with the participants, and a qualitative rich analysis of the interactions that were observed during the game sessions served as the basis of the findings. Where possible, the findings have been illustrated by verbatim quotations of the participants.

The main finding of the project was that there was evidence the participants were learning by doing and building their own mental models about what was happening. Also, all groups of participants (students, software developers, and project managers) improved their scores between the pre- and post-game surveys and this improvement was statistically significant.

The second finding was that games such as Simsoft are not sufficient learning vehicles by themselves and need to be supplemented by other methods. The software developers and project managers were able to make decisions based on experience or their university studies, but many students said they needed to know more than the game provided.

The third finding was that Simsoft is a suitable pedagogical device for participants of different skills and backgrounds. The participants in this research project came from a variety of Western and Eastern cultures; there were differences in language abilities; and experience in their fields ranged from nothing to seasoned professionals with a wide breadth of work and life experience. Yet, a majority of participants said they found the game interesting, it was pitched at the right level, and was something they could easily play and understand.

The fourth finding was that a majority of participants said they would be prepared to invest greater time and effort in games such as Simsoft if the reward was a deeper understanding of a problem domain. Many participants said the game ended too soon or that they would like to create a scenario similar to their own work place or that they wanted more time to discuss their decisions. A group of ten players had previously played a real-time stock market game and felt that games run in real time gave time for considered judgments and added verisimilitude.

The fifth finding was that the majority of the participants found working in groups was a positive experience. It has already been mentioned that the participants were a diverse group of cultures, skills, and experience and many felt they were able to work out collaborative decisions in a constructive manner. However, as with any group activity, facilitators need to be cognisant of any individuals dominating a group or others who might need a gentle prompt to contribute more.

The last finding was a majority of participants preferred to play around a game board rather than a fully computerised game because this was a familiar and simple activity and less time was lost to overcoming technological problems and to making simple ergonomic arrangements such as fitting all the team around a single computer. Even so, facilitators need to prepare the participants for the game sessions by giving clear instructions and sufficient lead time to absorb the information.

Once these findings had been derived, four participants (roughly 10% of the original research population as suggest by (Lincoln & Guba, 1984)) were chosen at random to check that the findings made overall sense in their experience of the game. Each of these participants was sent a URL to an online survey that presented the six findings above, and they were asked whether they agreed or disagreed. All concurred with the findings without comment.

Chapter 5– Analysis and Interpretation

Introduction

The purpose of this research project was to see if and how games could contribute to better software engineering management education by helping software engineers and project managers explore some of the dynamic complexities of the field in a safe and inexpensive environment. If games *could* contribute, then what features made them most efficacious?

The research was exploratory rather than deterministic and sought to understand a complex socio-technical system (software engineering management), therefore a qualitative research approach was used. Within this qualitative paradigm, grounded theory was used because it is a means of inductively developing a theory from the collected data.

To this end, a simple game, Simsoft, was developed and teams of post-graduate project management students (for software and general projects), and practising software project managers and developers played the game in teams in a series of game sessions. Overall game evaluation was made on the basis of pre- and post-game surveys, performance in Simsoft, and a qualitative rich analysis of the interactions that were observed during the game sessions.

The following research questions and associated hypotheses formed the basis of the project:

Can games contribute to better software engineering management education? Q1

Games built on sound software engineering management principles are a more effective means of improving software project management education than more traditional pedagogical means. H1

If games are to contribute to better software engineering management education, what features make them most efficacious? Q2

For best effect, players should be able to easily relate the context of a game to their real-world experience. H2.1

For best effect, games should be simple to play and understand and only as theoretically complex as needed to explore the H2.2

concepts at hand.

The findings in Chapter 4 largely support the research questions. The major finding of the research was that participants *were* learning as they played the game. So, games can contribute the better software project management education (research question *Q1*) by providing a safe and inexpensive environment in which to explore dynamic concepts. However, hypothesis *H1* was disproved as the findings suggest that games alone are not more effective than more traditional pedagogical means such as lectures, case studies, and readings.

The findings further suggested that simple games (hypothesis *H2.2*), and games in which the participants are able to relate game play to an external context (hypothesis *H2.1*), such as their real-world roles, are the most efficacious (research question *Q2*).

This chapter analyses and discusses the findings in more detail according to the following broad analytic categories:

- Games and learning (research question *Q1*).
- Games in context (research question *Q2*).
- The relative complexity of games (research question *Q2*).

These analytic categories are aligned to the research questions and were used to code the data that ultimately revealed the findings presented in the previous chapter. Whereas the findings were bald statements of what was found when the data from multiple sources was amalgamated, this chapter tries to answer the question, “What does it all mean?”.

This discussion takes into consideration related work in the field and how the results of this project support or deviate from these other efforts.

Analytic Category 1- Games and Learning

Learning in Simsoft

The first research question (*Q1*) was designed to see if games could contribute to software engineering management education and whether they were better than more traditional means of instruction.

The results showed that each group of participants (students, project managers, and software developers) improved their performance between the pre- and post-game tests. This suggests that the participants *were* constructing knowledge for themselves based on what they had experienced in the game. Comments from the participants supported this:

“Aha!”

“Now I see why”

When each group was further classified by years of experience in the field, the same improvement between the pre- and post-game tests was seen, with the greatest improvement being in those with more experience. For example, students gained relatively less from the game than more experienced software developers and project managers.

Together these results suggest that learning *is* happening, but for some participants at least some level of *a priori* knowledge is necessary to make more sense of what is happening in the game, which confirms that noticed by other researchers (Gredler, 1996, p. 36). (In an ideal problem-based learning environment, it is normal for participants to start an exercise with imperfect knowledge (Savin-Baden & Major, 2004, pp. 3 – 4), and then set about to resolve this dissonance. But, within the confines of a single-session game, this was not possible.) This, then, disproves hypothesis *H1* that games are a more effective means of improving software engineering management than other traditional pedagogical means. Instead, participants can learn some, but not all, of what they need to know from a game.

Learning Through Simsoft Compared to Others

The results that show learning is happening through Simsoft largely agree with those found by other researchers using games to teach aspects of the software development lifecycle (Caulfield et al., 2011e. See also Appendices L and M).

Appendix L: Full Data Extract of Games Used in Software Engineering Education shows that games have been used in a variety of ways to teach different aspects of

software engineering and software project management. In general, all those that assessed the degree of learning by the participants found that the participants were learning some new concepts or were reinforcing known theories. All the research projects, whether explicitly or implicitly stated, found that games alone were not sufficient pedagogical devices to teach software engineering or project management concepts and would have to be supplemented by other means. Only Navarro (2009) and Hainey et. al. (2010) evaluated the effectiveness of games for players of different skills and backgrounds and each found that games were suitable for a wide variety of participants. All these findings agree with those of this research project.

It should be noted, however, that apart from Navarro's and Drappa and Ludewig's body of work, many of the research projects in Appendix L had very small sample sizes and few were developed or repeated beyond that described in the initial papers. Also, the sample populations for all projects were under-graduate and post-graduate university students, so extrapolating the results to other populations can be problematic (Remus, 1986; Garb, 1989; Camerer & Johnson, 1991). In contrast, those that played Simsoft were a combination of students and experienced project managers and software developers. This, and other differences between Simsoft and similar research, will be discussed later.

Are Games More Effective Than Other Pedagogical Means?

Hypothesis *H1*— that games are a more effective means of improving software engineering management education than other means— turns out to be an artefact of a wider debate within education about the effectiveness of any one pedagogical device over any others. Some studies (Kulik et al., 1985; Kulik & Kulik, 1991) claim that computer-based instruction, for example, offers some advantage over other means, but the benefit is often so slight that ascribing it to anything in particular is risky (Clark, 1994). Others (Clark, 1994; Kozma, 1994; Tennyson, 1994) argue that one method can never be better than another. On reflection, this is not surprising:

If we were to chart out all the instructional topics, the wide variety of learners, and the many instructional situations, we would sometimes find an advantage for books, sometime teachers, sometimes film or video, sometimes peer-tutoring, sometimes hands-on field experience, sometimes listening to an audio tape, and sometimes computers. Not surprisingly, across these many studies, which utilized a variety of topics, learners, and

situations, little or no overall effect was found in favour of a single medium. (Alessi & Trollip, 2001, p. 6)

So, it makes sense to choose a pedagogical device based on what we know about the material, the students who are to receive this material, and the environment in which they will be learning. For games, certain scenarios lend themselves most:

- Where instruction by other means is prohibitively expensive, dangerous, or difficult. This includes, for example, military training (Perla, 1990; Riddell, 1997) and historical recreations (McCall, 2011). For the purposes of this research project, software engineering management education falls into this category: only in fiction (DeMarco, 1997) has a controlled experiment been conducted in which parallel development teams build the same product under different project conditions.
- Where the students are already acculturated to games or have been brought around to Kolb's (1984) experiential learning style by their teachers. Kolb imagined two dialectically opposed and intersecting dimensions: concrete experience/abstract conceptualisation and active experimentation/reflective observation. An effective learner must initially involve themselves fully, openly, and without bias in new experiences (concrete experience). They must then be able to reflect on and observe their experiences from many perspectives (reflective observation). From this they must be able to create concepts that integrate what they see and experience into logically sound theories (abstract conceptualisation). With this grounding, they must be able to then use these theories to make decisions and solve problems (active experimentation) (Kolb, 1984, p. 30).
- Where there are logistical difficulties such as catering for remote or distributed players.
- Where it is recognised that games can be more administratively demanding and resource intensive, particularly in the time required of teachers and facilitators than, say, traditional lectures (Watson & Blackstone, 1989, p. 493; Petranek, 1994; Bates & Poole, 2003, pp. 129 – 152).

A game may survive this assessment, be played, and yet still not deliver on its promise, which is ultimately transforming what has been learned in the game into reasoned action in the real world (Crookall & Thorngate, 2009; Kriz, 2009, p. 28). This may be because the game has been poorly designed or implemented; the latter

depends on the skill of the facilitator and the commitment of the players, but the former is of critical importance because everything derives from it and will be explored in more detail next.

Learning-Design Principles in Simsoft

In his seminal book on video games and education, *What Video Games Have to Teach Us About Learning and Literacy*, Gee (2007b) discusses 36 principles of learning he believes should be designed into every good game. Originally conceived for video games, and later condensed to 13 (Gee, 2007a), the principles parallel those found by other cognitive researchers (Bereiter & Scardamalia, 1993; diSessa, 2000; Gredler, 2004) and they have since been adopted for situations involving an active learner and any game. It is instructive to see how Simsoft addresses Gee’s principles (Table 10).

Table 10: Simsoft Compared with Gee's (2007a) Principles of Good Game Design

Learning Principle	In Simsoft
I. Empowered Users	
<p>1. Co-design: good learning means that players feel they are active agents (producers) not just passive recipients (consumers).</p> <p>In good games, players feel their actions and decisions– and not just those of the game designer– are co-designing the game world and the experiences they are having. It therefore matters what the player does because this determines a unique path through the game.</p>	<p>The course of game play in Simsoft is completely determined by the decisions the players make. They have full control of their workforce planning (subject to budget and timing restraints) and can increase or reduce hours as required.</p>
<p>2. Customise: different styles of learning work better for different people. People cannot be agents of their own learning if they cannot make decisions about how they learn best. At the same time, they should be able (and encouraged) to try new styles.</p> <p>Good games achieve this by naturally accommodating different styles of learning and playing or by allowing the players to customise the game play to fit their style.</p>	<p>Teams can organise themselves any way they wish. Some nominated a lead decision maker or arbiter, usually based on experience, while others were more collaborative and democratic.</p> <p>The game sessions contained enough time for the players to debate their decisions.</p>

<p>3. Identity: deep learning requires an extended commitment and such a commitment is typically created when people take on a new identity they value and in which they become heavily invested.</p> <p>Good games offer players identities in which they can rewardingly invest time and effort. This can be done by offering a character so intriguing that players want to inhabit the avatar and project onto it their own fantasies, desires, and pleasures. Alternatively, games may offer a relatively empty character upon which players can build a deep and consequential life history.</p>	<p>Players take on the role of a project manager– not something so exciting, particularly for experienced project managers. But a Simsoft project manager is unfettered by project politics and has complete control over the project's budget and workforce planning. This comment was from a project manager:</p> <p>“I wish I have [sic] this power at work”</p>
<p>4. Manipulation and distributed knowledge: cognitive research suggests perception and action are deeply interconnected. "Thus, fine-grained action at a distance - for example, when a person is manipulating a robot or watering a garden via a web cam - cause humans to feel as if their bodies and minds have stretched into a new space. More generally, humans feel expanded and empowered when they can manipulate powerful tools in intricate ways that extend their area of expertise."</p> <p>Good games almost always involve action at a (virtual) distance. The more intricately a player can control a character and objects in the game world, the more the player is willing to invest time and effort in the game.</p>	<p>The players had fine-grained control over their workforce, subject to budget constraints and hiring delays.</p>
<p>II. Problem Solving</p>	
<p>5. Well-ordered problems: problems in good games are designed so that the early challenges a player faces allows them to form good hypotheses they can use now and later.</p>	<p>Initially players made simple decisions about hiring more staff to ramp up the project. By the time they were confident with the mechanics of this process, the game state would have changed sufficiently so they would then have to make more complex decisions to balance work backlogs, the volume of rework, a looming deadline and reduced funds.</p>
<p>6. Pleasantly frustrating: learning works best</p>	<p>Simsoft demands more careful decisions as</p>

<p>when new challenges are pleasantly frustrating, that is at the outer edge of, but within, the player's regime of competence. These challenges feel hard, but doable. Players also need feedback so even if they fail, they have an idea of what must be done next time.</p>	<p>the game progresses. For example, the usual response to a large back log of work is to hire more staff, but the hiring delay means there is no immediate effect. A number of teams noticed this during the game:</p> <p>"We have to be careful about bringing on too many new hires. It'll ultimately clog things up".</p> <p>For all teams, the causal loop diagram on the back of the project briefing document was used to point out the counterintuitive nature of many project cycles.</p>
<p>7. Cycles of expertise: expertise in any field is created by repeated cycles of practice until the skills become nearly automatic. New skills are gradually added to the practice set and the cycle continues (Bereiter & Scardamalia, 1993). In games, we see this in the different levels a player must move through: there are cycles of extended practice, a test of mastery, then a new challenge which requires further extended practice. In this way the game moves forward at a predictable pace and the player senses achievement at each mastered skill.</p>	<p>More complex decisions need to be made as the game proceeds, but by this time the players will have mastered the mechanics of the game and the delays and counter-intuitive behaviour that are possible.</p> <p>Simsoft logs all game decisions so these can be studied or replayed.</p>
<p>8. Information should be delivered on demand and just in time: humans are not good at using information when it has little context and before they can practically use it. Instead, information is best used when it is given just in time (when it can be used straight away) and on demand (when there is a need to use it).</p>	<p>Each game session was preceded by a short briefing from the researcher about the mechanics of the game and then most sessions were under way within a couple of minutes. Each game schedule contained a causal loop diagram representing the underlying system dynamics model that players could refer to as needed in light the way pieces were moving on the board. The game board itself also shows the major work and personnel flows of the game.</p>
<p>9. Fish tanks: a fish tank can be a simple eco-system containing just a few controlled variables (water, light, food, fish). As such, it can show interactions between the variables that might otherwise be obscured in the real</p>	<p>Simsoft represents a simplified version of a software project: there are no requirements gathering, deployment, or maintenance phases. Instead, the game concentrates on a single, important factor– human</p>

<p>world. In a similar way, games are simplified systems that stress a few key variables and their interactions meaning players are not overwhelmed by the complexity of a whole system.</p>	<p>resources– without the noise these other phases may have introduced</p>
<p>10. Sandboxes: in games, as in the real world, sandboxes are safe, protected areas where things cannot go too wrong, too quickly and where any affects on the outside environment are minimised.</p> <p>In a good game, a sandbox may be a tutorial, or the first couple of levels may be sandboxed so that decisions made here do not completely spoil the player's chances later in the game.</p>	<p>Each game session was preceded by a short briefing from the researcher about how to make and enter game decisions. The range of initial decisions available was small so the players were able to see the flow of work over a number of project weeks before making more influential decisions.</p>
<p>11. Skills as strategies: there is a paradox in Principles 7 and 8: players need to practice certain skills in order to master them, but without a sufficient context, this practice may be seen as pointless.</p> <p>In good games, players learn and practice skills in order to accomplish specific things– they are a strategy for accomplishing something first, and of value as skills in themselves second.</p>	<p>The objective of Simsoft is the completion of the project within budget and on time. The skills the players are developing in the game are directly employed to this end.</p>
<p>III. Understanding</p>	
<p>12. Systems thinking: people learn new things (skills, strategies, and ideas) best when they see how these things fit into a larger system in which they have meaning.</p> <p>Good games help players understand how the simplified world of the game fits into a broader context, either of the game or of the real world.</p>	<p>While Simsoft only represents a slice of a real software development project, that slice sends ripples through most other areas of a typical project. This comment was from a software developer with 2 to 5 years experience:</p> <p>“I see my part in the machinery now”</p>
<p>13. Meaning as action image: humans do not usually think in abstract concepts and according to logical principles. Rather, we think through experiences we have had and then create imaginative reconstructions of that experience. To reason about, say, a football game we think about games we</p>	<p>For experienced software developers and project managers, thinking about their work in concrete rather than abstract terms is easy and connections can be made:</p> <p>“Now I see why”</p>

<p>have seen and heard about rather than generalities. For humans, words and abstract concepts have their deepest meanings when they are clearly tied to perception and action in the world.</p>	<p>“I hope that future versions will let me set up specific scenario and play them out. That would really help me in my work”</p> <p>For students, with less experience to draw on, meaning as action is harder to create. But, there are signs that experience in the game resonates: from a note scribbled on a game board beside the Rework box:</p> <p>“I must remember this”</p>
--	---

Sources: (Bereiter & Scardamalia, 1993; diSessa, 2000; Gee, 2004, 2007a, 2007b). (See also Caulfield et al., 2011d for how these design principles apply to software engineering curriculum guidelines)

A simple game like Simsoft cannot hope to fully address each of the above learning principles and call itself, in Gee’s loaded term, a good game, at least in its first iteration. Nevertheless, Simsoft comes close, if not for the tolerable parity demonstrated in Table 10, then only for the final comment against principle 13. A student was seen to scribble on a game board beside the Rework box, “I must remember this”. If Simsoft’s *raison d’être* is to allow software professionals to fail early and often in a place where failure is safe and can be learned from, then this comment shows that at least one person will be carrying a useful nugget of information into their next project.

Analytic Category 2– Games in Context

If games *could* be found to be useful in software engineering management education, then the second research question, *Q2*, sought to determine what features would make them most beneficial. Hypothesis *H2.1* posits one such feature: the context of the game should be something the players can easily relate to their real-world experience. Yet, software is often developed within an ill-structured environmental context (Bostrom & Heinen, 1977a, 1977b; Keen, 1981; Kling & Iacono, 1984; Hirschheim & Klein, 1989; Bennetts et al., 1998; Day, 2000) that includes sometimes contradictory human and business priorities. In this sort of environment, the best technical or logical solution is not always what the end user really wants.

Nevertheless, things can be done to impart game context. In the design of Simsoft, context refers to the objects of the game, the terminology used, and the basic work

flow— things that should be familiar to practising project managers and software engineers as well as students who have passed introductory courses. In practice, context came to mean the ways the participants wanted to play Simsoft and incorporate it into their work life, and which group gained more from the experience.

Context in Design

In Chapter 1 (see page 6), a game was defined as “a simulation that is purposefully run, wholly or partly determined by players’ decisions, within some predetermined circumstances”. While this definition is adequate for general purposes, it doesn’t describe the fundamental attributes of a game— the common language by which we can classify games. The most recent and accepted classification is that by Garris et al. (2002), who came up with six game dimensions by which any type of game could be classified:

- *Fantasy*. Games involve imaginary worlds, scenarios, and characters. The player uses their imagination to participate in unusual social situations and analogies of real-world processes in possibly unfamiliar locations. “Fantasies allow users to interact in situations that are not part of normal experience, yet they are insulated from real consequences” (Garris et al., 2002, p. 448).
- *Rule/Goals*. In a game, the confused and complex rules and constraints of normal life are put on hold and replaced by a precise and arbitrary set that are only operative within the fixed time and space of the game (Caillois, 1961, p. 7). Rules establish the criteria by which we can determine a winner.
- *Sensory stimuli*. Visual or auditory stimulations that help the player accept, for a time, they are participating in an alternate reality.
- *Challenge*. A Challenge represents “the ideal amount of difficulty and improbability of obtaining goals. A challenging game possesses multiple clearly-specified goals, progressive difficulty, and informational ambiguity. Challenge also adds fun and competition by creating barriers between current state and goal state” (Wilson et al., 2009, p. 230).
- *Mystery*. This represents a gap between what the player now knows and what they must know. This evokes the curiosity of the player because they want to resolve this dissonance. Mystery can be created by information incongruity, surprise and expectation violation, idea incompatibility, or incomplete or inconsistent information (Wilson et al., 2009, p. 231).

-
- *Control*. Control refers to the player's ability to exercise control over elements of the game. "Games evoke a sense of personal control when users are allowed to select strategies, manage the direction of activity, and make decisions that directly affect outcomes" (Garris et al., 2002, p. 451).

Of interest in assessing hypothesis *H2.1* are the dimensions of fantasy and rules/goals, which together constitute the context of a game.

The fantasy of Simsoft was established when players were given an instruction sheet that described the mechanics of game play and a simple statement of work about the project they were to manage (see Appendix D: Simsoft Instructions to Players). Most teams quickly entered into the spirit of the exercise by giving their team a name and deciding who would play which role. Some team names appeared to be carefully chosen to reflect the composition and camaraderie of the team: "United Nations" (each team members came from a different country); "NoBalls" (an all-female team); "The Convicts" (an all-Australian team); and "Sea Monkeys" (a team of weekend boat enthusiasts). When asked, most players said the project scenario was realistic and the accompanying instructions were easy to understand and follow.

The rules/goals dimension is governed by the System Dynamics model behind Simsoft. This model is based on the project work structure first discussed and modelled by Roberts (1964). Roberts' initial work has been expanded greatly by subsequent researchers who have added rich details based on actual projects (see Lyneis & Ford, 2007 for a comprehensive survey of the field), but the underlying work structure remains unchanged. The work-to-do, review, rework, work-completed cycle on the left-hand side of the Simsoft game board follows this structure. These boxes mimic the product backlog or sprint backlog, rework, and burndown artefacts of an agile project (Martin, 2002a; Schwaber, 2004; Cockburn, 2006). In fact, one team labelled their game board as such (Figure 12).

The right-hand side of the Simsoft game board represents the human resources of the project **Error! Reference source not found.** and depicts how developers increase in experience, and hence productivity and value, over time (Brooks, 1995; DeMarco & Lister, 1999). Because this is well-known in the field, a number of teams recognised the pattern:

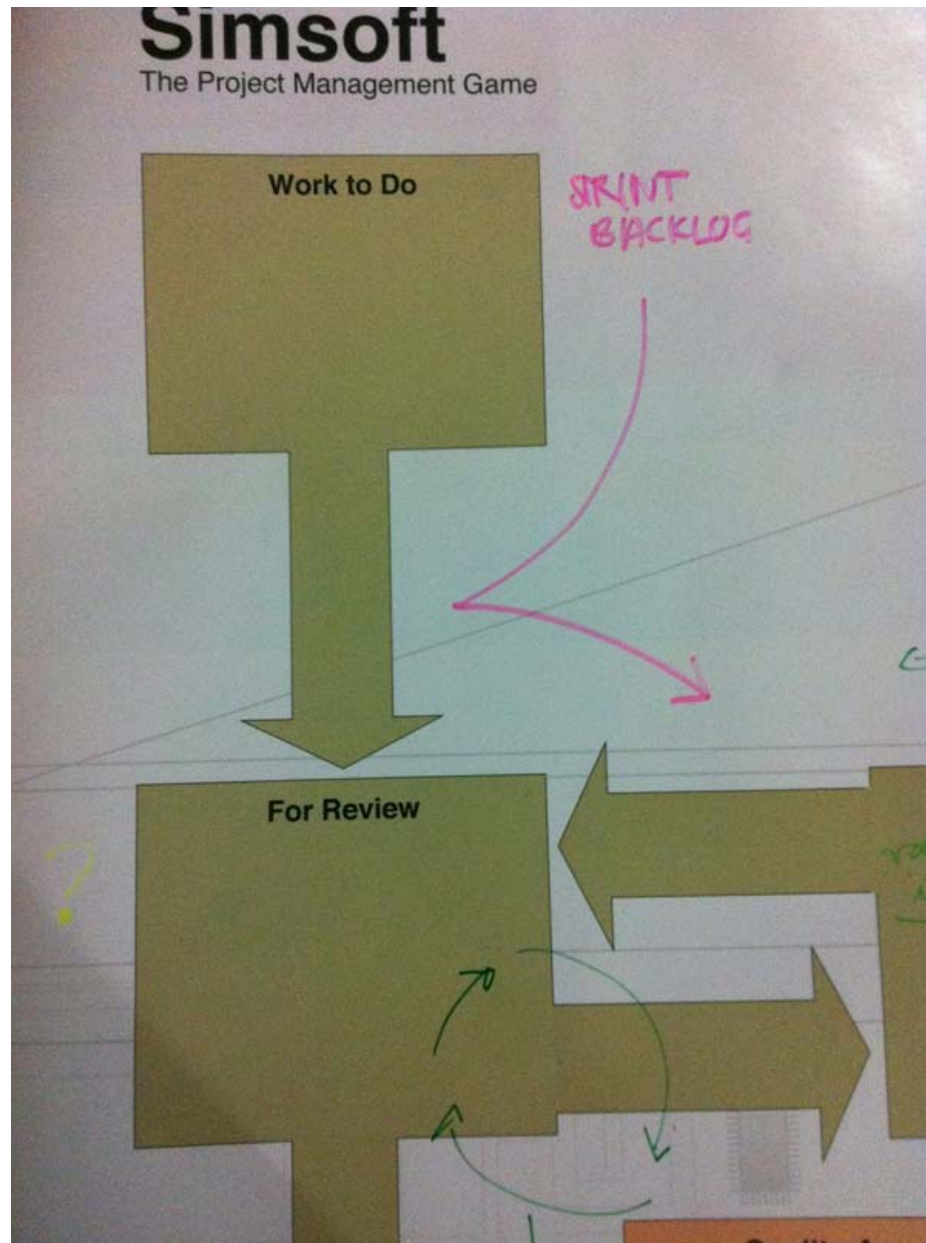


Figure 12: Simsoft game board marked-up during a game session with an agile development term.

“[One of our team] had read about Brooks’ model and could let us know if we were on the right track”

Together, these behaviours suggest that even though the participants were being asked to accept an alternate reality governed by simple but arbitrary rules, the context created by Simsoft was not so dissociated from reality that the participants weren’t able to draw parallels with familiar things.

Context in Practice

The database of Simsoft game sessions showed that games lasted for an average of 35 minutes and that 80% of games finished within 40 minutes. Afterwards, the participants were encouraged to stay and discuss their performance with other teams. Often, these after-game sessions lasted longer than the game sessions themselves as the participants talked about strategies and what had worked or failed for their project.

A common comment during these after-game gatherings, and something that was reflected in the post-game survey, was that most participants were prepared to invest greater time and effort in games such as Simsoft if the reward was deeper understanding of the problem domain.

With this in mind, one participant suggested running the game in real time, so that one week of real time equated to one week of project time. During the week, the team members could do research and discuss their options before coming to a carefully considered decision about their next step. This suggestion was influenced by a stock market game a number of participants had played the previous year, in which teams bought and sold shares on a fantasy stock exchange based on real prices published in the daily newspaper. The winner after three months was the team with the largest portfolio. In the week between submitting buy and sell orders, the players researched likely companies, scanned market reports, and took note of interest rate decisions, the price of oil and gold, and currency fluctuations to see how they might affect the market.

This suggestion represents a desire to put Simsoft *more* in context, by allowing the participants to step out of the fantasy world of the game, do some research, and then step back into the game with better knowledge. However, Simsoft, and all other games discovered during a systematic survey of the literature (Appendix L), are played in one-off sessions. What players learned, had to be learned within the hour or so of the game session. Of course, games could be replayed, but they must have sufficient depth to present alternate, engaging paths through the game in repeat. For even the most sophisticated game in the field, SimSE, players became bored when playing second and subsequent times (Navarro & van der Hoek, 2007, p. 5).

For some participants, an extended game session with breaks is necessary. Evaluation of the pre- and post-game scores showed that students gained relatively less from the game than more experienced project managers and developers (see Finding 2 for more details). The following comment from a student is illustrative:

“I saw in the game aspects of theory covered at uni, but without knowing the theory first I probably wouldn’t have recognised the significance.”

That is, students in this research population didn’t have the *a priori* knowledge needed to make full sense of the game’s dynamics.

Playing the game over multiple, rather than single, sessions would more closely conform to the tenets of Problem-Based Learning where participants begin their project with imperfect knowledge and then have to identify and learn what they needed in order to solve the issue at hand.

No Game is an Island

Hypothesis *H2.1*— that players should be able to easily relate the context of the game to their real-world experience— is analogous with the opening lines of John Donne’s (Craik & Craik, 1986) well-known poem:

No man is an island entire of itself; every man
is a piece of the continent, a part of the main;

In a similar way, *H2.1* says that games are part of a whole, or a context, for those who play them. This whole encompasses the working environment the players come from, the real-world experience the players bring to the game, and the game experience the players take back to their working environment.

How a game is designed can help create this context by setting a conceit (the fantasy and rules of the game) that is familiar to the participants. They may be taking on new roles and performing unfamiliar tasks, but objects of the game, the terminology used, and the basic work flow must create a recognisable simulacrum. Simsoft participants demonstrated they were able to draw parallels between the game world and their real world.

Context can also be created in the way the game is played. The single-session format of Simsoft and other games means participants make decisions based on what they know now, rather than truly informed decisions. Playing games in more depth and across multiple sessions would give the participants the opportunity to actively relate the game world to their real world.

Together, these features and behaviours support hypothesis H2.1.

Analytic Category 3- The Relative Complexity of Games

In contrast to most current software project games, Simsoft is relatively simple: it uses a large game board to show the flow and the current state of a project and it concentrates on the slim, but important, build phase of the development lifecycle. This design flows from hypothesis *H2.2* which posits that games should only be as complex as absolutely necessary to explore the concepts at hand.

When asked, most Simsoft players agreed with this hypothesis and said they preferred a board game to a fully computerised version because they could start playing more quickly without having to learn how to navigate a new user interface and without fear of making an unintended move. For example:

“Sometimes technology gets in the way”

“Everyone plays board games so we all know what to do”

Apart from the mechanics of playing Simsoft, the simple design meant the state of the game and its underlying causal model were always visible:

“The board game [was] simple and I could easily see the state of the game”

The appeal of simplicity over complexity has been noted before. While complex games offer “the richest learning experience available, the game’s very formidable appearance probably intimidated a number of players or forced them into a learning situation they were unprepared or unwilling to negotiate” (Wolfe, 1978, p. 152).

The next most effective game in Wolfe’s study was found to be the least complex, supporting similar research that showed relatively simple games can provide essentially the same, if not more, benefits as the more complex (Raia, 1966, p. 351;

Watt, 1977, p. 2; Butler et al., 1979; Meadows, 1999; Dempsey et al., 2002). Therefore, making games only as complex as absolutely necessary, or hiding unnecessary detail, could be a way of achieving the best learning outcomes while avoiding the player mortality (boredom and dropout) noted by Wolfe.

Compared to a computer game, the simple Simsoft board game offered another advantage: they were the scratch pads where players could write notes and reminders and hints as they played the game (Figure 13). Four teams asked for copies of their graffitied game boards

A board game also more easily fosters the collaboration needed in any team enterprise such as a software development project. When a computer or online game is played by multiple participants, likely at different physical locations, the basic cues of identity, personality, and body language are hidden. Without these cues, researchers have found that many computer games explicitly designed to be collaborative will degenerate into competitive games at worst or games in which “everyone just kind of does their own thing” (Zagal et al., 2006, p. 25) at best.

In Simsoft, group play was viewed positively by most participants. It reflected real-world experience and also meant ideas and opinions could be shared:

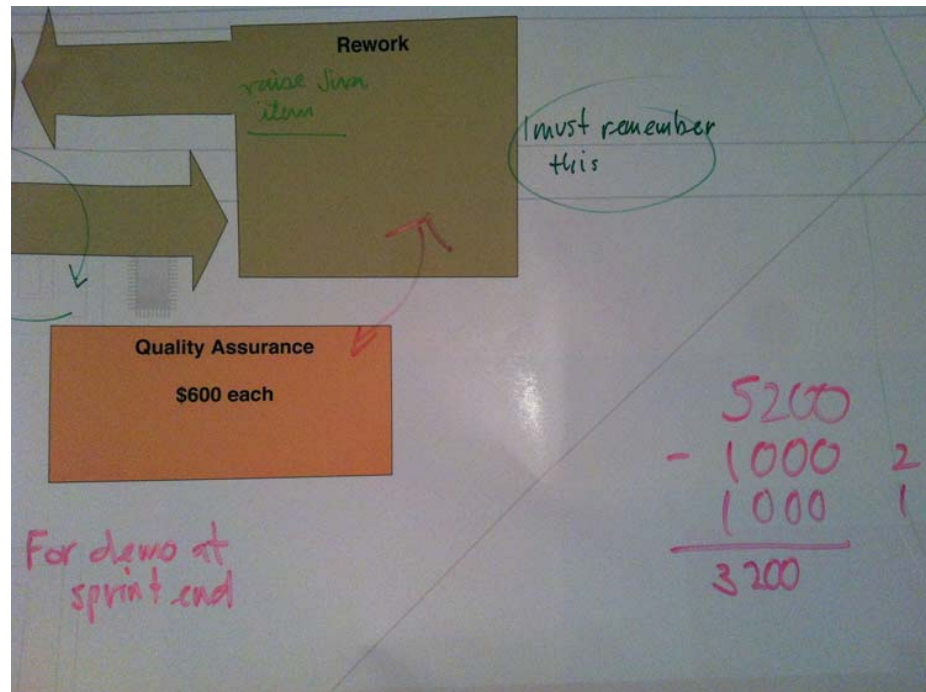


Figure 13: A section of one the game boards marked-up with players' notes and reminders.

“It was like [the agile] stand up meeting we have every morning”

“I thought software development was a solitary experience but it's not really”

“Everyone had a chance to offer an opinion”

Notwithstanding these positive aspects, any group activity may devolve into groupthink (Janis, 1971) in which the opinion of a dominant individual or clique prevails, possibly against reasonable evidence. In the Simsoft game sessions, no teams were larger than four participants and many participants were known to each other, either professionally or socially, so there was ample opportunity to contribute to the discussions or even dispute the idea of a colleague or friend. There were also no more than four game sessions running at once, which meant the researcher was able to notice any participants standing back and gently prompt them for a suggestion.

Few other software development game researchers have looked closely at these same aspects of game design. Hainey et al. (2010) asked players to rate game features such as graphics, realism of the characters, realism of the environment, and sound, but these were evaluations of the verisimilitude of these features, not their appropriateness to the task at hand. On this same rating of game features, collaboration ranked last or second last across all players, but this is to be expected in a single-player game. Similarly, other researchers (Baker et al., 2005; Navarro & van der Hoek, 2009; Zapata, 2010) asked their participants if they enjoyed playing the game or whether they found it engaging, but these questions asked the participants to evaluate a particular game's representation of its environment rather than its comparative complexity or its value as a collaborative tool.

Related Work

Recalling the definitions of model, simulation, and game given in Chapter 1:

A model is a convenient representation (in words, numbers, or other symbols) of some real-world socio-economic or socio-technical system; a simulation is a dynamic, operational model through which changes over time are revealed; and a game is a simulation that is purposefully run,

wholly or partly determined by players' decisions, within some predetermined circumstances.

It can be said that software development has been *modelled* (Belady & Lehman, 1976; McCabe, 1976; Remus & Zilles, 1979; Boehm, 1981) and *simulated* (Abdel-Hamid & Madnick, 1991; Variale et al., 1994; Hansen, 1996; Madachy, 1996; Tvedt, 1996; Collofello, 2000; Martin & Raffo, 2001) many times. But, these are not the software development perspectives of interest for this research project because:

- They focus primarily on predicting rather than educating. For example, Boehm's COCOMO model (2000) is designed to calculate the cost and effort of a software project based on historical data and what is currently known about the project at hand. COCOMO is used to validate an estimate, not necessarily find out why it is this number.
- They are not interactive or designed for group participation. For example, perhaps the most well-known simulation (Abdel-Hamid & Madnick, 1991) contains over 300 underlying variables, but doesn't have a way to interact with the model except through direct manipulation of these variables at a source code level (Martin, 2002b, pp. 32 - 37).

Given their focus, it is not surprising that these models and simulations fail most, if not all, of Gee's principles of interactive game design (see Table 10). In contrast, Appendix L details a number of other research projects that have used games — and more closely align with Gee's principles— in some role in software project management or education. Still, there are differences between these games and Simsoft.

SimSE, the game developed by Navarro (2009) and her colleagues at the University of California, Irvine over a number of years, is perhaps the most advanced game in the field and the only one in Appendix L that has been developed much beyond its initial implementation. SimSE supports a number of different development approaches (such as rapid prototyping, code inspection, and the Rational Unified Process), provides users with a performance report after they complete the game, and has also been tested and verified in a range of controlled classroom settings. Players manage their SimSE project through a rich graphical user interface that shows their team at work, along with various management

reports and dials. In contrast to Simsoft, SimSE is a single-user game so without players clustering around a single screen, there's little opportunity to discuss and debate project decisions and come to a consensus. SimSE is also heavily focussed on the process of software development– the *how* of software development– whereas Simsoft is also concerned with the *who*.

Like Simsoft, a number of the games in the field have eschewed computers, either completely or partly, in favour of playing cards, boards, and sometimes dice. For example, in Zapata's (2010) game, teams throw a dice, that determines which of a collection of technical questions the team must answer. From here, the team gets a chance to estimate the size of a project component and score points. This slightly convoluted game show format relies more on chance than skill and means that most players are dormant and passive while other teams are having their turn. Chance also plays a role in games like *Problems and Programmers* (Baker et al., 2005)– players draw cards from a shuffled deck– and *PlayScrum* (Fernandes & Sousa, 2010)– a roll of the dice determines what resources the player can accumulate and what problems may be encountered. Unlike Simsoft, these games are competitive rather than co-operative.

Some of the games in Appendix L operate at a very high level with players performing broad project functions. As a result they see only general project dynamics. In SimVBSE (Jain & Boehm, 2006), SimjavaSP (Shaw & Dermoudy, 2005), MO-SEProcess (Zhu et al., 2007), Hainey's game (2010), and OSS (Sharp & Hall, 2000) players make their avatar visit certain rooms or characters to ask questions or collect information. In Hainey's game the result of this office tour is a requirements document that is then passed to the project manager avatar for assessment. The tour may have to be repeated if all the requirements haven't been identified. A game interface makes this engaging for a while, but how it relates to real-world software engineering management is dubious. Providing the same information in a short project description, such as the one that comes with Simsoft, means the player can begin exploring the problem domain sooner. And, with less effort required to create the office environment, more could be devoted to the interesting detail of the project's dynamics.

SESAM (Drappa & Ludewig, 1999; Drappa & Ludewig, 2000) could almost be called a model or simulation rather than a game because a user runs it by typing commands in a complex modelling language and the system responds in kind. In

exchange for this complexity, SESAM allows its users to define a wide variety of development methodologies as well as hire and fire staff, assign tasks, and ask developers about their progress. But, without an effective visual interface, playing SESAM is like programming an old VCR: there isn't enough feedback to know if the instructions have been entered correctly or what is happening as a result (Norman, 1988, pp. 51 – 53). It is perhaps not surprising that SESAM has not been developed far beyond that described in the original papers. In contrast, Simsoft's state of play is always visible on the game board.

One feature common to all the research projects in Appendix L is the population they use: the participants are either undergraduate or post-graduate university, and in one case high school, students. In broader research circles, there is some debate (Remus, 1986; Garb, 1989; Camerer & Johnson, 1991) about whether students make viable candidates for research involving management decisions because they may lack the experience and knowledge to make their responses transferable to the workplace. The research population of this project was a mixture of university students and project managers and software developers of varying lengths of experience, making this transfer easier.

In summary, there are four main differences between the approach taken in this research project and others in the area:

- Simsoft is equally, if not more, concerned with *who* does the work in a software development as it is with process of *how* the work is done. This echoes the cover of Boehm's (Boehm, 1981) *Software Engineering Economics* which shows personnel is where the greatest productivity gains are possible.
- Simsoft is a board game (with a small calculator component) in contrast to other games that use a graphical user interface of varying levels of richness. Often the user interface is simply a conceit of the game for performing housekeeping functions and lends little to the real purpose— a common mistake made by many game designers (Crawford, 2003, pp. 114 – 115). Other games that use playing cards or games boards contain an element of chance rather than skill.
- Simsoft is cast at a level of detail at which the players can see the movement of individual pieces of work and individuals themselves. Games cast at higher levels, such as OSS, mask fundamental project dynamics.

- The research sample for this project is a mixture of students and experienced professionals rather than wholly students.

Summary

This chapter has analysed the findings from a series of game sessions to try to determine if and how games might contribute to better software engineering management education. The analysis has shown that games *can* contribute and has also pointed to features that can be built into games, and implementation techniques, that make games more efficacious.

Reviewing the research questions and hypotheses that formed the basis of this research project, one hypothesis was disproved and two were proved:

<i>Can games contribute to better software engineering management education?</i>	<i>Q1</i>	
<i>Games built on sound software project management principles are a more effective means of improving software project management education than more traditional pedagogical means.</i>	<i>H1</i>	<i>Disproved</i>
<i>If games are to contribute to better software engineering management education, what features make them most efficacious?</i>	<i>Q2</i>	
<i>For best effect, players should be able to easily relate the context of a game to their real-world experience.</i>	<i>H2.1</i>	<i>Proved</i>
<i>For best effect, games should be simple to play and understand and only as theoretically complex as needed to explore the concepts at hand.</i>	<i>H2.2</i>	<i>Proved</i>

For the first research question, the analysis of the findings showed that participants were learning the things Simsoft was designed to teach, meaning games *can* play a role in software project management education. However, games alone are not sufficient to teach all that a well-rounded software professional needs to know, disproving hypothesis *H1*. Ideally, games like Simsoft should be used in conjunction with other teaching devices such as lectures, case studies, and

readings; and used once a foundation of software engineering and project management theory has been laid. Games could therefore be a way of bridging the gaps between a degree's requisite courses and the final, important capstone project.

The second research question considered the features of software engineering management games that might make them most useful as teaching tools. A review of the literature suggested that existing games in the field were unnecessarily complex and lacked a sufficient context to allow players to easily apply their real world knowledge in the game and, in turn, apply experience gained in the game in the real world. The analysis of the findings suggested that:

- Games which established a context familiar to the participants helped them apply their experience to the game scenario and they were also able to relate experiences in the game to real-world activities, proving hypothesis *H2.1*.
- Simple games could deliver effective learning outcomes, proving hypothesis *H2.2*.

These analyses are largely consistent with those of others in the field. Games have shown themselves to be effective teaching tools in software engineering management, but as supplementary rather than primary devices. Most of the participants in this research project said they were keen to spend more time and effort on games, but only if games could reveal more of what they wanted to know.

In addition, the particular design of this research project produced some findings that were new. Simsoft was designed to be played by groups of people around a large game board, whereas many existing games in the field are single-user computer games. This proved to be a positive experience for most participants because they were able to share their experiences and negotiate a group decision on how to proceed— keys skills in a modern collaborative workforce.

The analysis of the findings presented here should be read mindful of the limitations described in Chapter 3 (see page 78). This analysis applies to a particular population of students, project managers, and software developers playing a particular type of game— a confluence of factors that may not recur often. In mitigation, the descriptions of Simsoft, the game sessions, and the participants' responses have been made as rich and as detailed as possible so that readers can best gauge what might be relevant for their own circumstances.

Chapter 6– Conclusions and Recommendations

Introduction

The purpose of this research project was to see if and how games could contribute to better software engineering management education by helping software engineers and project managers explore some of the dynamic complexities of the field in a safe and inexpensive environment. If games *could* contribute, then what features made them most efficacious? The conclusions from this study follow this theme and, drawing on the findings, address four main areas:

- The value of games in software engineering management education— should we be playing games?
- Long-form games as a way of creating context.
- Games as group activities.
- Simple games can be effective.

The following is a discussion of the conclusions drawn from this research, followed by the researcher’s recommendations, and a final reflection on the study.

Shall We Play a Game?

In the 1983 movie, *War Games*, a young Matthew Broderick plays David Lightman, a hacker who has broken into WOPR– the War Operation Plan Response supercomputer which is programmed to play out different doomsday scenarios and learn from them so it can eventually take full, automated control of the United States’ nuclear arsenal. When David is presented with a screen prompt that asks, “Shall we play a game?”, he innocently selects “Global Thermonuclear War”. As quickly becomes apparent, WOPR is ready to do more than just play games and it starts executing commands in readiness for a *real* missile strike against the Soviet Union.

The portentous question asked by WOPR– shall we play a game?– had meaning for this research project too, but without the same dire consequences. In effect: shall we– should we– play games in software project management education? The answer, we believe, is a qualified, *yes*. The answer is qualified because our findings show that while games are useful pedagogical tools and are well-received by

players, they are not sufficient in themselves and must be supplemented by other learning devices.

Findings 1 (the participants are learning by doing), 2 (games are not sufficient learning vehicles by themselves), and 3 (games are suitable pedagogical devices for participants of different skills and backgrounds) show that games are useful for reinforcing known theories and for teaching new concepts, and can be used in mixed groups, but they need to be supplemented by other teaching means. If participants don't have some foundation knowledge of software engineering and project management concepts, they may miss some of what the game is trying to teach.

What a game is trying to teach must also be aforesought for game designers. A game that simply mimics, for example, a software engineering project will not necessarily teach its players much. When the learning objectives are clearly understood, they can be catered for in specific design attributes.

Long-Form Games as a Way of Creating Context

The fourth finding of this research project was that most participants were prepared to invest greater time and effort in games like Simsoft if the reward was a deeper understanding of the problem domain. The conclusion to be drawn from this is that when learning is engaging, immersive, relevant, and even fun, people are prepared to try it in long-form.

Many of the existing games in the field are played in single sessions usually lasting about an hour. In these short-form games, what learning there is, must be imparted in this hour or so. A long-form game, by comparison, would be played over multiple sessions across a number of days or weeks or even in real time as suggested by some Simsoft participants. Between game sessions, the participants could meet to discuss their options, do research into relevant areas, and generally reflect on how the game context agrees or disagrees with what they know. How they interpret these similarities and resolve the differences is key to how much they will take away from games such as Simsoft.

Games as Group Activities

It is easy to appreciate why most of the current games in the field of software engineering management education have been implemented as single-user, computer games:

- The computations of the game can be performed quickly and accurately.
- The games offer a rich interface which likely appeals to game-literate players.
- The games can be easily distributed to many participants.

But, for all these advantages, some key benefits are forsaken. When games are played by single users sitting before a computer, the normal cues of personality, identity, and body language of people working in a group are missing. Of course, these cues could be represented through game avatars, but the psychology of character personality and its graphical implementation would require an effort that would dwarf the central purpose of games such as those considered here (Rousseau & Hayes-Roth, 1996; Baylor, 2011). This effort might still be a shadow of what one might encounter around a simple game board.

The fifth finding of this project found that most participants saw working in groups as a positive experience because it reflected normal work place practice and they were able to share ideas and draw on a wide variety of experience. From this we can conclude that it is possible for games to teach beyond their original remit when they serve as a means of bringing people together to learn from each other (Prensky, 2006, pp. 96 – 100).

Simple Games Can Be Effective

The final finding of this research project was that the majority of participants preferred playing a board game to a fully computerised game because it was simple, easy to play, and clearly demonstrated the concepts at hand. The conclusion we can draw from this is that simple games can be as effective as more complex games. Because simple games are naturally pared down to basics, they can be developed and deployed more quickly than their richer, more complex counterparts and yet still deliver many of the same benefits.

Recommendations

Based on the findings, analysis, and conclusions of this research project, some recommendations for educators and trainers, game designers, and further research can be offered.

Recommendations for Educators and Trainers

Educators and trainers should consider:

- Implementing serious games such as Simsoft as a supplementary component of software engineering and software project management training. For best effect, the games should be run late in a semester when students have built up the necessary background knowledge, or run at any time if the players are known to be experienced. An ideal place within the curriculum to play games before the final, synthesising capstone project.
- Playing the game sessions collaboratively, with the participants working in small groups. The participants should be encouraged to use the game boards as work spaces where they can write down thoughts and notes and other information to be shared with their team. Facilitators of these sessions need to be aware of groupthink and will need to quietly monitor the game play to make sure everyone has an opportunity to contribute.

Recommendations for Game Developers

Those developing new games or reworking existing games should consider:

- Articulating the learning objectives of the game and tracing these to specific design features of the finished product. Without this, learning will be haphazard at best, and won't occur at worst.
- Developing games in long-form, multi-session formats in which the game scenario is revealed or developed over time. In between, the players should be given an opportunity to research the domain problem and meet with their team members to discuss what they have discovered.
- Developing games that promote collaboration and interaction over competition and single-player mode. Computer and on-line games don't yet do this well, so board games are a practical alternative.
- Balancing the time and effort required to create rich, highly-configurable games with that required to create simple, specific games. The shorter build-play-rework life cycle of the latter means that games can be put to use quickly and reworked if necessary without great cost, and yet still be effective.

Recommendations for Future Research

This research project is the first implementation of Simsoft and naturally the sample size of those who have played it and offered their feedback and insights is still relatively small. A natural path for future research is to run more game sessions and see if the results obtained here and now hold true for a larger population and in other places.

Beyond this, the results of this project suggest some interesting possibilities for future research:

- A further similar study should be undertaken using a more complex game than Simsoft to fully test the worth of simple versus complex games.
- A further study should be undertaken that runs Simsoft over multiple game sessions, rather than as a single session, as was recommended by a number of participants. Presently, the game state is saved after each move, which means the game board can be reconstructed at any point in time, however the scenario would need to be resigned to suit this new format.
- On-line and computer games have some attractive advantages, but they give up the serendipity of people from different backgrounds and with different experiences coming together around a game board to solve a problem. However, social networking software and web sites offer an intriguing opportunity to create a distributed learning game that may be able to replicate the benefits of collaborative, personal interactions.
- Simsoft looks specifically at human resources aspects of the build phase of a software engineering project, but much happens before (such as requirements gathering and design) and after (such as testing and maintenance) this phase. Expanding Simsoft into these areas will increase the fidelity of the game and will be an opportunity to see if, for example, Brooks' Law (Brooks, 1995; Caulfield & Maj, 2002), holds for other phases of the development life cycle.
- Is the management of a software project that much different from the management of, say, a hardware project or a construction project? To answer this, Simsoft should be played by participants from other disciplines to see if the results obtained here can be applied to other domains.
- It was gratifying to note that other besides students seem to gain greater understanding from playing Simsoft. Exploring this further was outside the

problem statement of this project, but the use of games in post-tertiary and on-the-job training deserves further consideration.

Final Reflections

This research project is now coming to a close: long checked-out books are being returned to the library, papers are being filed, computer files organised and archived, desks are being cleared. This seems to be an appropriate place to reflect on what we know now that we didn't know at the start.

In many fields, games are mature tools for instruction and research. War games, for example, have a lineage stretching back many thousands of years, and business games have been informing, frustrating, and delighting managers, students, and researchers since the 1950s. Games in software engineering management are a relatively new addition to the field, but already there are some exemplars and some abandoned paths. Perhaps Simsoft will join the former in time.

Underlying the two research questions on which Simsoft is based is a higher-order question, mentioned here for the first time and originally posed by Herbert Simon (Simon, 1996): can a simulacrum, such as a game, ever tell us anything that we do not already know? Taking for granted the plausible assertion that a simulacrum can be no better than the assumptions built into it, then the answer to Simon's question might be, *no*.

Yet, Simon argues otherwise. For example, even if we start with correct premises, it may be hard to infer meaning because humans generally have trouble mentally tracing through even a small set of causal relationships and then making sense of the result. A disinterested game is better able to do and show this because it makes our mental models visible so they can be critiqued by ourselves and others. It may also be the case that in some circumstances, for some players, games may only tell us what we do not already know:

... games are more than a caricature of life; they are an introduction to life— an introduction to the idea of rules, which are imposed on all alike, an introduction to the idea of playing under different sets of rules— that is, the idea of different roles, an introduction to the idea of aiding another person and of knowing that one can expect aid from another, an introduction to the idea of working toward a collective goal and investing oneself in a collectivity larger than oneself (Coleman, 1975a).

We respectfully leave it to others to decide whether Simsoft and the research presented here is a similar introduction.

References

- Abdel-Hamid, T. K. and Madnick, S. E. (1983). 'The Dynamics of Software Project Scheduling.' *Communications of the ACM*, vol. 26, no. 5 (May), pp. 340 – 346.
- Abdel-Hamid, T. K. and Madnick, S. E. (1991). *Software Project Dynamics: An Integrated Approach*. Englewood Cliffs: Prentice-Hall
- Abt, C. C. (1970). *Serious Games*. New York: The Viking Press
- Aldrich, C. (2005). *Learning by Doing: A Comprehensive Guide to Simulations, Computer Games, and Pedagogy in e-Learning and Other Educational Experiences* San Francisco: Pfeiffer
- Aldrich, C. (2009). *The Complete Guide to Simulations and Serious Games: How the Most Valuable Content Will be Created in the Age Beyond Gutenberg to Google*. New York: Pfeiffer
- Alessi, S. M. and Trollip, S. R. (2001). *Multimedia for Learning: Methods and Development*, 3rd edition. Boston: Allyn and Bacon
- Alluisi, E. A. (1991). 'The Development of Technology for Collective Training: SIMNET, A Case History.' *Human Factors*, vol. 33, no. 3 (June), pp. 343 – 362.
- Amador, J. A., Miles, L. and Peters, C. B. (2007). *The Practice of Problem-Based Learning: A Guide to Implementing PBL in the College Classroom*. Boston: Anker Publishing Company
- Ambler, S. W. (2006). 'Supersize Me.' *Software Development*, vol. 14, no. 3 (March), pp. 46 – 48.
- Amstutz, A. E. (1963). 'Management Games— A Potential Perverted.' *Industrial Management Review*, vol. 5, no. 1 (Fall), pp. 29 – 36.
- Anderson, P. (1999). 'Complexity Theory and Organization Science.' *Organization Science*, vol. 10, no. 3 (May - June), pp. 216 – 232.
- Anderson, R. S. and Puckett, J. B. (2003). 'Assessing Students' Problem-Solving Assignments.' In D. S. Knowlton and D. C. Sharp (eds.), *Problem-Based Learning in the Information Age*, pp. 81 – 87. San Francisco: Jossey-Bass.
- Andlinger, G. R. (1958a). 'Business Games— Play One!' *Harvard Business Review*, vol. 36, no. 2 (March - April), pp. 115 – 125.
- Andlinger, G. R. (1958b). 'Looking Around: What Can Business Games Do?' *Harvard Business Review*, vol. 36, no. 4 (July - August), pp. 147 – 160.
- Applegate, L. M., Montealegre, R. and Knoop, C.-I. (1996a). *BAE Automated Systems (B): Implementing the Denver International Airport Baggage-Handling System* (Case study 9-396-312). Boston: Harvard Business School.
- Applegate, L. M., Montealegre, R., Nelson, H. J. and Knoop, C.-I. (1996b). *BAE Automated Systems (A): Denver International Airport Baggage-Handling System* (Case study 9-396-311). Boston: Harvard Business School.
- Ardis, M. A., Chenoweth, S. V. and Young, F. H. (2008). 'The "Soft" Topics in Software Engineering Education.' *Proceedings of the 38th ASEE/IEEE Frontiers in*

-
- Education Conference*, (Saratoga Springs, 22-25 October 2008), pp. F3H-1-F3H-6.
- Armarego, J. (2002). 'Advanced Software Design: A Case in Problem-Based Learning.' *Proceedings of the 15th Conference on Software Engineering Education and Training (CSEE&T 2002)* (Covington, Kentucky, 25 - 27 February), pp. 44 - 54. Los Alamitos: IEEE Computer Society.
- Ashby, W. R. (1956). 'Self-Regulation and Requisite Variety.' In F. E. Emery (ed.) *Systems Thinking*, (1972), pp. 105 - 124. Ringwood: Penguin Books.
- Awad, E. M. and Ghaziri, H. M. (2008). *Knowledge Management*. Delhi: Dorling Kindersley
- Babb, E. M., Leslie, M. A. and Van Syke, M. D. (1966). 'The Potential of Business Gaming Methods in Research.' *The Journal of Business*, vol. 39, no. 4 (October), pp. 465 - 472.
- Baber, R. L. (1982). *Software Reflected: The Socially Responsible Programming of Our Computers*. Amsterdam: North-Holland Publishing Company
- Baker, A., Navarro, E. O. and van der, H., Andre (2003). 'Problems and Programmers: An Educational Software Engineering Card Game.' *Proceedings of the 25th International Conference on Software Engineering*, (Portland, Oregon, 3 - 10 May, 2003), p. 614. Los Alamitos: IEEE Computer Society.
- Baker, A., Oh Navarro, E. and van der Hoek, A. (2005). 'An Experimental Card Game for Teaching Software Engineering Processes.' *The Journal of Systems and Software*, vol. 75, no. 1 - 2, pp. 3 - 16.
- Baker, A. C., Jensen, P. J. and Kolb, D. A. (1997). 'In Conversation: Transforming Experience into Learning.' *Simulation & Gaming*, vol. 28, no. 1 (March), pp. 6 - 12.
- Barell, J. (2006). *Problem-Based Learning: An Inquiry Approach*, 2nd edition. Thousand Oaks: Corwin Press
- Barlas, S. (1996a). 'Anatomy of a Runaway: What Grounded the AAS.' *IEEE Software*, vol. 13, no. 1 (January), pp. 104 - 106.
- Barlas, S. (1996b). 'FAA Shifts Focus to Scaled-Back DSR.' *IEEE Software*, vol. 13, no. 2 (March), pp. 110, 114.
- Barros, M. d. O., Dantas, A. R., Veronese, G. O. and Werner, C. M. L. (2006). 'Model-Driven Game Development: Experience and Model Enhancements in Software Project Management Education.' *Software Process: Improvement and Practice*, vol. 11, no. 4, pp. 411 - 421.
- Barrows, H. S. and Tamblyn, R. (1976). 'An Evaluation of Problem-Based Learning in Small Groups Utilizing a Simulated Patient.' *Journal of Medical Education*, vol. 51, no. 1, pp. 52 - 54.
- Barrows, H. S. and Tamblyn, R. (1977). 'The Portable Patient Problem Pack: A Problem-Based Learning Unit.' *Journal of Medical Education*, vol. 52, no. 12, pp. 1002 - 1004.
- Bates, A. W. and Poole, G. (2003). *Effective Teaching with Technology in Higher Education*. San Francisco: Jossey-Bass

-
- Baylor, A. (2011). 'The Design of Motivational Agents and Avatars.' *Educational Technology Research and Development*, vol. 59, no. 2, pp. 291 – 300.
- Becker, K. (2011). 'The Magic Bullet: A Tool for Assessing the Evaluating Learning Potential in Games.' *International Journal of Game-Based Learning* vol. 1, no. 1 (January - March), pp. 19 - 31.
- Belady, L. A. and Lehman, M. M. (1976). 'A Model of Large Program Development.' *IBM Systems Journal*, vol. 15, no. 3, pp. 225 – 252.
- Bennetts, P. D. C., Wood-Harper, A. T. and Mills, S. (1998). 'The Soft System Methodology as a Framework for Software Process Improvement.' *Journal of End User Computing*, vol. 10, no. 1 (Winter), pp. 12 – 19.
- Bereiter, C. and Scardamalia, M. (1993). *Surpassing Ourselves: An Inquiry into the Nature and Implications of Expertise*. Chicago: Open Court
- Bloom, B. S., Masia, B. B. and Krathwohl, D. R. (1956). *Taxonomy of Educational Objectives: The Classification of Educational Goals, Handbook I: Cognitive Domain*. London: Longman
- Bloomberg, L., Dale and Volpe, M. (2008). *Completing Your Qualitative Dissertation*. Thousand Oaks: Sage Publications
- Boehm, B., Egyed, A., Port, D., Shah, A., Kwan, J. and Madachy, R. (1998). 'A Stakeholder Win-Win Approach to Software Engineering Education.' *Annals of Software Engineering*, vol. 6, no. 1, pp. 295 - 321.
- Boehm, B. and Turner, R. (2003). *Balancing Agility and Discipline*. Boston: Addison-Wesley
- Boehm, B. W. (1981). *Software Engineering Economics*. Sydney: Prentice-Hall
- Boehm, B. W., Abts, C., Brown, A. W., Chulani, S., Clark, B. K., Horowitz, E., Madachy, R. J., Reifer, D. J. and Steece, B. (2000). *Software Cost Estimation with Cocomo II*. Upper Saddle River: Prentice Hall
- Boguslaw, R. and Pelton, W. J. (1959). 'STEPS, A Management Game for Programming Supervisors.' *Datamation*, vol. 5, no. 6 (November – December), pp. 13 – 16.
- Boocock, S. S. (1970). 'Using Simulation Games in College Courses.' *Simulation & Games*, vol. 1, no. 1 (March), pp. 67 – 79.
- Booker, E. (1994). 'Have You Driven a Simulated Ford Lately?' *Computerworld*, vol. 28, no. 27 (4 July), p. 76.
- Bostrom, R. P. and Heinen, J. S. (1977a). 'MIS Problems and Failures: A Socio-Technical Perspective. Part I: The Causes.' *MIS Quarterly*, vol. 1, no. 3 (September), pp. 17 – 32.
- Bostrom, R. P. and Heinen, J. S. (1977b). 'MIS Problems and Failures: A Socio-Technical Perspective. Part II: The Application of Socio-Technical Theory.' *MIS Quarterly*, vol. 1, no. 4 (December), pp. 11 – 28.
- Botkin, J. W., Elmandjra, M. and Malitza, M. (1979). *No Limits to Learning: Bridging the Human Gap: A Report to the Club of Rome*. Oxford: Pergamon Press
- Bots, P. and van Daalen, E. (2007). 'Functional Design of Games to Support Natural Resource Management Policy Development.' *Simulation & Gaming*, vol. 38, no. 4 (December), pp. 512 – 532.

-
- Boulding, K. E. (1964). *The Meaning of the Twentieth Century: The Great Transition*. New York: Harper & Row
- Bourque, P., Dupuis, R., Abran, A., Moore, J. W. and Tripp, L. (1999). 'The Guide to the Software Engineering Body of Knowledge.' *IEEE Software*, vol. 16, no. 6 (November/December), pp. 35 - 44.
- Bradley, J. and McGrath, G. M. (2000). 'Boot Camp or Bordello: Whipping Rookies into Shape.' *Proceedings of the Twenty First International Conference on Information Systems*, (Brisbane), pp. 467 – 472. Atlanta: Association for Information Systems.
- Bredemeier, M. E. and Greenblat, C. S. (1981). 'The Educational Effectiveness of Simulation Games.' *Simulation & Gaming*, vol. 12, no. 3 (September 1, 1981), pp. 307 – 332.
- Brereton, O. P., Lees, S., Bedson, R., Boldyreff, C., Drummond, S., Layzell, P. J., Macaulay, L. A. and Young, R. (2000). 'Student Group Work Across Universities: A Case Study in Software Engineering.' *IEEE Transactions on Education*, vol. 43, no. 4 (November), pp. 394 – 399.
- Bridges, E. M. (1992). *Problem Based Learning for Administrators*. Eugene: ERIC
- Brooks, F. P. (1987). 'No Silver Bullet: Essence and Accidents of Software Engineering.' *IEEE Computer*, vol. 20, no. 4 (April), pp. 10 – 19.
- Brooks, F. P. (1995). *The Mythical Man-Month: Essays on Software Engineering*, 20th anniversary. Sydney: Addison-Wesley. (1975)
- Bryant, A. (2000). 'It's Engineering Jim... but not as we know it.' *Proceedings of the 22nd International Conference on Software Engineering*, (Limerick, Ireland), pp. 77 – 86. Los Alamitos: IEEE Computer Society Press.
- Buckley, W. (1968). 'Society as a Complex Adaptive System.' In W. Buckley (ed.) *Modern Systems Research for the Behavioral Scientist: A Sourcebook*, pp. 490 – 513. Chicago: Aldine Publishing Co.
- Burgess, T. F. (1991). 'The Use of Computerized Management and Business Simulation in the United Kingdom.' *Simulation & Gaming*, vol. 22, no. 2 (June), pp. 174 – 195.
- Buschmann, F. (2011a). 'Gardening Your Architecture, Part 1: Refactoring.' *IEEE Software*, vol. 28, no. 4 (July/August), pp. 92 – 94.
- Buschmann, F. (2011b). 'Gardening Your Architecture, Part 2: Reengineering and Rewriting.' *IEEE Software*, vol. 28, no. 5 (September/October), pp. 21 – 23.
- Butler, R. J., Pray, T. F. and Strang, D. R. (1979). 'An Extension of Wolfe's Study of Simulation Game Complexity.' *Decision Sciences*, vol. 10, (July), pp. 480 – 486.
- Buxton, J. N. and Randell, B., (eds.). (1970). *Software Engineering Techniques: Report on a Conference Sponsored by the NATO Science Committee, Rome, Italy, 27th to 31st October 1969*. Brussels: Scientific Affairs Division, NATO.
- Caillois, R. (1961). *Man, Play and Games*. Translated by M. Barash. New York: Free Press of Glencoe. (Original publication 1958)
- Camerer, C. F. and Johnson, E. J. (1991). 'The Process-Performance Paradox in Expert Judgment.' In K. A. Ericsson and J. Smith (eds.), *Toward a General Theory of Expertise: Prospects and Limits*, pp. 195 – 217. Cambridge: Cambridge University Press.

-
- Card, O. S. (1985). *Ender's Game*. New York: Tom Doherty Associates
- Caulfield, C. W. (2001). *A Case for Systems Thinking and System Dynamics*. Unpublished Masters thesis, Edith Cowan University, Perth, Western Australia.
- Caulfield, C. W. (2005). 'Regular Expressions and RPG.' *iSeries NEWS*, no. 309 (July), pp. PROVIP19 – 25.
- Caulfield, C. W. (2006a). 'Open Source Build Management for Java Projects– Part 1.' *Sys Admin*, vol. 15, no. 8 (August), pp. 37 – 45.
- Caulfield, C. W. (2006b). 'Open Source Build Management for Java Projects– Part 2.' *Sys Admin*, vol. 15, no. 9 (September), pp. 42 – 48.
- Caulfield, C. W. (2009a). 'Use Code Inspection Tools to Improve Your Java Code.' *System i NEWS*, no. 352 (February), pp. PROVIP17 – PROVIP21.
- Caulfield, C. W. (2009b). 'Use Sun SPOTs as Your Build Canary.' [on-line]. Available at <http://www.ibm.com/developerworks/java/library/j-spots/index.html>. Last accessed
- Caulfield, C. W., Kohli, G. and Maj, S. P. (2004). 'Sociology in Software Engineering.' *Proceedings of the 2004 American Society for Engineering Education Annual Conference & Exposition* (Salt Lake City). Washington: American Society for Engineering Education.
- Caulfield, C. W. and Maj, S. P. (2001). 'A Case for Systems Thinking and System Dynamics.' *Proceedings of the 2001 IEEE International Conference on Systems, Man & Cybernetics*, (Tucson, Arizona, 7 - 10 October), pp. 2793 - 2798. Piscataway, New Jersey: IEEE Computer Society.
- Caulfield, C. W. and Maj, S. P. (2002). 'A Case for System Dynamics.' *Global Journal of Engineering Education*, vol. 6, no. 1, pp. 25 – 34.
- Caulfield, C. W. and Maj, S. P. (2007). 'Come Play.' In M. Iskander (ed.) *Innovative Techniques in Instruction Technology, E-Learning, E-Assessment, and Education*, pp. 86 – 91. New York: Springer.
- Caulfield, C. W., Maj, S. P., Xia, J. and Veal, D. (2011a). 'Shall We Play a Game?' *Modern Applied Science*, vol. in press.
- Caulfield, C. W., Veal, D. and Maj, S. P. (2011b). 'Implementing System Dynamics Models in Java.' *International Journal of Computer Science and Network Security* vol. 11, no. 7 (July), pp. 43 – 49.
- Caulfield, C. W., Veal, D. and Maj, S. P. (2011c). 'Teaching Software Engineering Management – Issues and Perspectives.' *International Journal of Computer Science and Network Security* vol. 11, no. 7 (July), pp. 50 – 54.
- Caulfield, C. W., Veal, D. and Maj, S. P. (2011d). 'Teaching Software Engineering Project Management – A Novel Approach for Software Engineering Programs.' *Modern Applied Science*, vol. 5, no. 5 (October), pp. 87 – 104.
- Caulfield, C. W., Xia, J., Veal, D. and Maj, S. P. (2011e). 'A Systematic Survey of Games Used for Software Engineering Education.' *Modern Applied Science*, vol. 5, no. 6, pp. 28 – 43.
- Charness, G., Rigotti, L. and Rustichini, A. (2007). 'Individual Behavior and Group Membership.' *The American Economic Review*, vol. 97, no. 4, pp. 1340 – 1352.

-
- Cheng, Y.-P. and Lin, J. M.-C. (2010). 'A Constrained and Guided Approach for Managing Software Engineering Course Projects.' *IEEE Transactions on Education*, vol. 53, no. 3 (August), pp. 430 – 436.
- Cherryholmes, C. H. (1966). 'Some Current Research on Effectiveness of Educational Simulations: Implications for Alternative Strategies.' *American Behavioral Scientist*, vol. 10, no. 2 (October), pp. 4 – 7.
- Clark, R. R. (1994). 'Media Will Never Influence Learning.' *Educational Technology Research and Development*, vol. 42, no. 2, pp. 21 – 29.
- Cockburn, A. (2006). *Agile Software Development: The Cooperative Game*, 2nd edition. Boston: Addison-Wesley
- Cohen, K. J., Cyert, R. M., Dill, W. R., Kuehn, A. A., Miller, M. H., Van Wormer, T. A. and Winters, P. R. (1960). 'The Carnegie Tech Management Game.' *The Journal of Business*, vol. 33, no. 4 (October), pp. 303 – 321.
- Cohen, K. J. and Rhenman, E. (1961). 'The Role of Management Games in Education and Research.' *Management Science*, vol. 7, no. 2, pp. 131 – 166.
- Coleman, J. S. (1975a). 'In Defense of Games.' In C. S. Greenblat and R. D. Duke (eds.), *Gaming-Simulation: Rationale, Design and Applications*, pp. 72 – 74. New York: Sage Publications.
- Coleman, J. S. (1975b). 'Social Processes and Social Simulation Games.' In C. S. Greenblat and R. D. Duke (eds.), *Gaming-Simulation: Rationale, Design and Applications*, pp. 130 – 147. New York: Sage Publications.
- Collofello, J. (2000). 'University/Industry Collaboration in Developing a Simulation Based Software Project Management Training Course.' In S. A. Mengel and P. J. Knoke (eds.), *Proceedings of the Thirteenth Conference on Software Engineering Education & Training*, (Austin, Texas), pp. 161 – 168. Los Alamitos: IEEE Computer Society Press.
- Conan Doyle, A. (1974). 'A Scandal in Bohemia.' *The Adventures of Sherlock Holmes*, (Original publication 1892), pp. 15 – 40. London: Pan Books.
- Connolly, T. M., Stansfield, M. and Hainey, T. (2007). 'An Application of Games-Based Learning within Software Engineering.' *British Journal of Educational Technology*, vol. 38, no. 3, pp. 416 – 428.
- Constantine, L. L. (1995). *Constantine on Peopleware*. Englewood Cliffs: Yourdon Press
- Copeland, D. G., Mason, R. O. and McKenney, J. L. (1995). 'Sabre: The Development of Information-Based Competence and Execution of Information-Based Competition.' *IEEE Annals of the History of Computing*, vol. 17, no. 3, pp. 30 – 56.
- Costikyan, G. (2005). 'I Have No Words & I Must Design.' In K. Salen and E. Zimmerman (eds.), *The Game Design Reader*, pp. 192 – 211. Cambridge, Massachusetts: The MIT Press.
- Courtois, P. J. (1985). 'On Time and Space Decomposition of Complex Structures.' *Communications of the ACM*, vol. 28, no. 6 (June), pp. 590 – 603.
- Cox, B. J. (1990). 'There Is a Silver Bullet.' *Byte*, vol. 15, no. 10 (October), pp. 209 – 218.
- Craik, T. W. and Craik, R. J., (eds.). (1986). *John Donne: Selected Poetry and Prose*. London: Methuen.

-
- Crawford, C. (2003). *On Game Design*. Indianapolis: New Riders Publishing
- Crawford, W. (2001). 'Y2K: Lessons from a Non-Event.' *Online*, vol. 25, no. 2 (March/April), pp. 73 – 74.
- Creswell, J. W. (2009). *Research Design: Qualitative, Quantitative, and Mixed Methods Approaches*, 3rd edition. Thousand Oaks: Sage Publications
- Crookall, D. (2010). 'Serious Games, Debriefing, and Simulation/Gaming as a Discipline.' *Simulation & Gaming*, vol. 41, no. 6 (December), pp. 898 – 920.
- Crookall, D. and Thorngate, W. (2009). 'Acting, Knowing, Learning, Simulating, Gaming.' *Simulation & Gaming*, vol. 40, no. 1 (February), pp. 8 – 26.
- Dale, A. G. and Klasson, C. R. (1962). *Business Gaming: A Survey of American Collegiate Schools of Business*. Austin: Bureau of Business Research, University of Texas
- Dangerfield, B. and Roberts, C. (1995). 'Projecting Dynamic Behavior in the Absence of a Model: An Experiment.' *System Dynamics Review*, vol. 11, no. 2 (Summer), pp. 157 – 172.
- Dantas, A. R., Barros, M. d. O. and Werner, C. M. L. (2004). 'A Simulation-Based Game for Project Management Experiential Learning.' In F. Maurer and G. Ruhe (eds.), *Proceedings of the 16th International Conference on Software Engineering & Knowledge Engineering*, (Banff, Alberta, Canada, 20 - 24 June), pp. 19 – 24. Los Alamitos: IEEE Computer Society.
- Davis, A. (1993). 'Software Lemmingengineering.' *IEEE Software*, vol. 10, no. 5 (September), pp. 79 – 81, 84.
- Dawson, R. J., Newsham, R. W. and Fernley, B. W. (1997). 'Bringing the 'Real World' of Software Engineering to University Undergraduate Courses.' *IEE Proceedings - Software Engineering*, vol. 144, no. 5 – 6 (October – December), pp. 287 – 290.
- Dawson, R. J., Newsham, R. W. and Kerridge, R. S. (1992). 'Introducing New Software Engineering Students to the 'Real World' at the GPT Company.' *Software Engineering Journal*, vol. 7, no. 3 (May), pp. 171 – 176.
- Day, J. (2000). 'Software Development as Organizational Conversation: Analogy as a Systems Intervention.' *Systems Research and Behavioral Science*, vol. 17, no. 4 (July - August), pp. 349 – 358.
- de Geus, A. P. (1992). 'Modelling to Predict or to Learn?' *European Journal of Operational Research*, vol. 59, no. 1, pp. 1 – 5.
- Defense Science Board (2002). *Defense Science and Technology*. Washington: Defense Science Board.
- DeMarco, T. (1982). *Controlling Software Projects*. New York: Yourdon Press
- DeMarco, T. (1991). 'Non-Technological Issues in Software Engineering.' *Proceedings of the 13th International Conference on Software Engineering*, (Austin, Texas), pp. 149 – 150. Los Alamitos: IEEE Computer Society Press.
- DeMarco, T. (1997). *The Deadline: A Novel About Project Management*. New York: Dorset House Publishing
- DeMarco, T. and Lister, T. (1999). *Peopleware: Productive Projects and Teams*, 2nd edition. New York: Dorset House Publishing Co

-
- Dempsey, J., Lucassen, B., Gilley, W. and Rasmussen, K. (1994). 'Since Malone's Theory of Intrinsically Motivating Instruction: What's the Score in the Gaming Literature.' *Journal of Educational Technology Systems*, vol. 22, no. 2, pp. 173 – 183.
- Dempsey, J. V., Haynes, L. L., Lucassen, B. A. and Casey, M. S. (2002). 'Forty Simple Computer Games and What They Could Mean to Educators' *Simulation & Gaming*, vol. 33, no. 2, pp. 157 – 168.
- Denning, P. J. and Riehle, R. D. (2009). 'The Profession of IT: Is Software Engineering Engineering?' *Communications of the ACM*, vol. 52, no. 3 (March), pp. 24 – 26.
- Denzin, N. K. (1989). *Interpretive Interactionism*. Newbury Park: Sage Publications
- Dewey, J. (1938/1963). *Experience and Education*. Kappa Delta Pi Lecture Series. New York: Collier Books
- Dill, W. R. and Doppelt, N. (1963). 'The Acquisition of Experience in a Complex Management Game.' *Management Science*, vol. 10, no. 1 (October), pp. 30 – 46.
- diSessa, A. A. (2000). *Changing Minds: Computers, Learning, and Literacy*. Cambridge, Massachusetts: The MIT Press
- Dooley, K. (1996). 'A Nominal Definition of Complex Adaptive Systems.' *Chaos Network*, vol. 8, no. 1, pp. 2 – 3.
- Dorn, D. S. (1989). 'Simulation Games: One More Tool on the Pedagogical Shelf.' *Teaching Sociology*, vol. 17, no. 1, pp. 1 – 18.
- Drappa, A. and Ludewig, J. (1999). 'Quantitative Modeling for the Interactive Simulation of Software Project.' *The Journal of Systems and Software*, vol. 46, no. 15 April, pp. 113 – 122.
- Drappa, A. and Ludewig, J. (2000). 'Simulation in Software Engineering Training.' *Proceedings of the 22nd International Conference on Software Engineering*, (Limerick, Ireland), pp. 199 – 208. New York: ACM Press.
- Duch, B. J., Groh, S. E. and Allen, D. E., (eds.). (2001). *The Power of Problem-Based Learning*. Sterling, Virginia: Stylus Publishing.
- Duke, R. D. (1980). 'A Paradigm for Game Design.' *Simulation & Games*, vol. 11, no. 3 (September), pp. 364 – 377.
- Eldredge, D. L. and Watson, H. J. (1996). 'An Ongoing Study of the Practice of Simulation in Industry.' *Simulation & Gaming*, vol. 27, no. 3 (September), pp. 375 – 386.
- Ellington, H. I. (1995). 'The Future of Simulation/Gaming in Britain.' In D. Crookall and K. Arai (eds.), *Simulation and Gaming Across Disciplines and Cultures: ISAGA at a Watershed*, pp. 225 – 229. Thousand Oaks: Sage Publications.
- Emam, K. E. and Koru, A. G. (2008). 'A Replicated Survey of IT Software Project Failures.' *IEEE Software*, vol. 25, no. 5, pp. 84 – 90.
- Eveleens, J. L. and Verhoef, C. (2010). 'The Rise and Fall of the Chaos Report Figures.' *IEEE Software*, vol. 27, no. 1 (January/February), pp. 30 – 36.
- Faria, A. J. (1987). 'A Survey of the Use of Business Games in Academia and Business.' *Simulation & Games*, vol. 18, no. 2 (June), pp. 207 – 224.
- Faria, A. J. (1998). 'Business Simulation Games: Current Usage Levels—An Update.' *Simulation & Gaming*, vol. 29, no. 3 (September), pp. 295 – 308.

-
- Faria, A. J. and Wellington, W. J. (2004). 'A Survey of Simulation Game Users, Former-Users, and Never-Users.' *Simulation & Gaming*, vol. 35, no. 2 (June), pp. 178 – 207.
- Feldman, H. D. (1995). 'Computer-Based Simulation Games: A Viable Educational Technique for Entrepreneurship Classes?' *Simulation & Gaming*, vol. 26, no. 3 (September), pp. 346 – 360.
- Fernandes, J. M. and Sousa, S. M. (2010). 'PlayScrum - A Card Game to Learn the Scrum Agile Method.' *Proceedings of the 2010 Second International Conference on Games and Virtual Worlds for Serious Applications*. Washington: IEEE Computer Society.
- Flexner, A. (1910). *Medical Education in the United States and Canada*. New York: The Carnegie Foundation for the Advancement of Teaching
- Ford, A. (1999). *Modeling the Environment: An Introduction to System Dynamics Modeling of Environmental Systems*. Washington: Island Press
- Ford, G. and Gibbs, N. E. (1996). *A Mature Profession of Software Engineering*. Pittsburgh: Software Engineering Institute, Carnegie Mellon University.
- Forrester, J. W. (1961). *Industrial Dynamics*. Waltham: Pegasus Communications
- Forrester, J. W. (1969). *Urban Dynamics*. Portland: Productivity Press
- Forrester, J. W. (1971). *World Dynamics*. Portland: Productivity Press
- Forrester, J. W. (1975). 'Counterintuitive Behavior of Social Systems.' *Collected Papers of Jay W. Forrester*, pp. 211 – 237. Cambridge, Massachusetts: MIT Press.
- Frezza, S. T., Tang, M.-H. and Brinkman, B. J. (2006). 'Creating an Accreditable Software Engineering Bachelor's Program.' *IEEE Software*, vol. 23, no. 6 (November/December), pp. 27 – 35.
- Fullford, D. A. (1996). 'Distributed Interactive Simulation: It's Past, Present, and Future.' In J. M. Charnes, D. J. Morrice, D. T. Brunner and J. J. Swain (eds.), *Proceedings of the 1996 Winter Simulation Conference*, (Coronado, California, United States), pp. 179 – 185. New York: ACM Press.
- Fulmer, R. M. (1993). 'The Tools of Anticipatory Learning.' *Journal of Management Development*, vol. 12, no. 6, pp. 7 – 14.
- Garb, H. N. (1989). 'Clinical Judgment, Clinical Training, and Professional Experience.' *Psychological Bulletin*, vol. 105, no. 3, pp. 387 – 396.
- Garris, R., Ahlers, R. and Driskell, J. E. (2002). 'Games, Motivation, and Learning: A Research and Practice Model.' *Simulation & Gaming*, vol. 33, no. 4 (December), pp. 441 – 467.
- Gee, J. P. (2004). *Situated Language and Learning: A Critique of Traditional Schooling*. London: Routledge
- Gee, J. P. (2007a). *Good Video Games and Good Learning: Collected Essays on Video Games, Learning and Literacy*. New York: Peter Lang Publishing
- Gee, J. P. (2007b). *What Video Games Have to Teach Us About Learning and Literacy*. New York: Palgrave MacMillan

-
- Gibbs, W. W. (1994). 'Software's Chronic Crisis.' *Scientific American*, vol. 271, no. 3 (September), pp. 86 – 95.
- Glass, R. L. (1998). *Software Runaways*. Upper Saddle River: Prentice Hall
- Glass, R. L. (1999). *Computing Calamities: Lessons Learned from Products, Projects, and Companies That Failed*. Upper Saddle River: Prentice Hall
- Glass, R. L. (2000). 'Y2K and Believing in Software Practice.' *Communications of the ACM*, vol. 43, no. 3 (March), pp. 17 – 18.
- Glass, R. L. (2006). 'The Standish Report: Does It Really Describe a Software Crisis?' *Communications of the ACM*, vol. 49, no. 8 (August), pp. 15 – 16.
- Goodwin, J. S. and Franklin, S. G. (1994). 'The Beer Distribution Game: Using Simulation to Teach Systems Thinking.' *Journal of Management Development*, vol. 13, no. 8, pp. 7 – 15.
- Gosen, J. and Washbush, J. (2004). 'A Review of Scholarship on Assessing Experiential Learning Effectiveness.' *Simulation & Gaming*, vol. 35, no. 2, pp. 270 – 293.
- Gredler, M. E. (1996). 'Educational Games and Simulations: A Technology in Search of a Research Paradigm.' In D. H. Johassen (ed.) *Handbook of Research on Educational Communications and Technology: A Project of the Association for Educational Communications and Technology*, pp. 521 – 539. New York: MacMillan.
- Gredler, M. E. (2004). 'Games and Simulations and Their Relationships to Learning.' In D. H. Jonassen (ed.) *Handbook of Research on Educational Communications and Technology*, 2nd edition, pp. 571 – 581. Mahwah, New Jersey: Lawrence Erlbaum Associates Publishers.
- Greenblat, C. S. (1973). 'Teaching with Simulation Games: A Review of Claims and Evidence.' *Teaching Sociology*, vol. 1, no. 1, pp. 62 – 83.
- Greenlaw, P. S. and Wyman, F. P. (1973). 'The Teaching Effectiveness of Games in Collegiate Business Courses.' *Simulation & Games*, vol. 4, no. 3 (September), pp. 259 – 294.
- Gresse von Wangenheim, C. and Shull, F. (2009). 'To Game or Not to Game?' *IEEE Software*, vol. 26, no. 2 (March/April), pp. 92 – 94.
- Gresse von Wangenheim, C., Thiry, M. and Kochanski, D. (2009). 'Empirical Evaluation of an Educational Game on Software Measurement.' *Empirical Software Engineering*, vol. 14, no. 4, pp. 418 – 452.
- Guba, E. G. and Lincoln, Y. S. (2005). 'Paradigmatic Controversies, Contradictions, and Emerging Confluences.' In N. K. Denzin and Y. S. Lincoln (eds.), *The Sage Handbook of Qualitative Research*, 3rd edition, pp. 191 – 215. Thousand Oaks: Sage Publications.
- Guetzkow, H. (1959). 'A Use of Simulation in the Study of Inter-Nation Relations.' *Behavioral Science*, vol. 4, pp. 183 – 191.
- Hacker, D. J. and Dunlosky, J. (2003). 'Not All Matacognition is Created Equal.' In D. S. Knowlton and D. C. Sharp (eds.), *Problem-Based Learning in the Information Age*, pp. 73 – 79. San Francisco: Jossey-Bass.

-
- Hainey, T., Connelly, T. J., Stansfield, M. and Boyle, E. A. (2010). 'Evaluation of a Game to Teach Requirements Collection and Analysis in Software Engineering at Tertiary Education Level.' *Computers & Education*, vol. 56, no. 1, pp. 21 – 35.
- Hand, H. H. and Sims, H. P. (1975). 'Statistical Evaluation of Complex Gaming Performance.' *Management Science*, vol. 21, no. 6 (February), pp. 708 – 717.
- Hansen, G. A. (1996). 'Simulating the Software Development Process.' *IEEE Computer*, vol. 29, no. 1 (January), pp. 73 – 77.
- Hart, R. J. and Sulzen, R. H. (1988). 'Comparing Success Rates in Simulated Combat: Intelligent Tactics vs. Force.' *Armed Forces and Society*, vol. 14, no. 2 (Winter), pp. 273 – 286.
- Hays, R. T. (1989). *Simulation Fidelity in Training System Design: Bridging the Gap Between Reality and Training*. New York: Springer-Verlag
- Hazzan, O. (2010). 'Putting Human Aspects of Software Engineering in University Curricula.' *IEEE Software*, vol. 27, no. 4 (July/August), pp. 90 – 91.
- Heldman, K. (2007). *PMP: Project Management Professional Exam Study Guide*, 4th edition. San Francisco: Sybex
- Hermann, C. F. (1967). 'Validation Problems in Games and Simulations with Special Reference to Models of International Politics.' *Behavioral Science*, vol. 12, pp. 216 – 231.
- Herz, B. and Merz, W. (1998). 'Experiential Learning and the Effectiveness of Economic Simulation Games.' *Simulation & Gaming*, vol. 29, no. 2 (June), pp. 238 – 250.
- Highsmith, J. A. (2000). *Adaptive Software Development: A Collaborative Approach to Managing Complex Systems*. New York: Dorset House
- Hirschheim, R. and Klein, H. K. (1989). 'Four Paradigms of Information Systems Development.' *Communications of the ACM*, vol. 32, no. 10 (October), pp. 1199 – 1216.
- Hobsons (2011). *The Good Universities Guide to Universities & Private Higher Education Providers*
- Homer, J. B. (1985). 'Worker Burnout: A Dynamic Model with Implications for Prevention and Control.' *System Dynamics Review*, vol. 1, no. 1, pp. 42 – 62.
- Horning, J. J. and Wortman, D. B. (1977). 'Software Hut: A Computer Program Engineering Project in the Form of a Game.' *IEEE Transactions on Software Engineering*, vol. 3, no. 4, pp. 325 – 330.
- Huizinga, J. (1971). *Homo Ludens*. Boston: Beacon Press. (Original publication 1950)
- Humphrey, W. S. (1998). 'Why Don't They Practice What We Preach?' *Annals of Software Engineering*, vol. 6, no. 1, pp. 201-222.
- Hung, D., Chen, V. and Lim, S. (2009). 'Unpacking the Hidden Efficacies of Learning in Productive Failure.' *Learning Inquiry*, vol. 3, no. 1, pp. 1 – 19.
- Hung, W., Bailey, J. H. and Jonassen, D. H. (2003). 'Exploring the Tensions of Problem-Based Learning: Insights from Research.' In D. S. Knowlton and D. C. Sharp (eds.), *Problem-Based Learning in the Information Age*, pp. 13 – 23. San Francisco: Jossey-Bass.

-
- Hunt, A. and Thomas, D. (1999). *The Pragmatic Programmer: From Journeyman to Master*. Boston: Addison-Wesley
- IBM (2011). *SPSS Statistics* [computer software], version 19.0. Available from <http://www.spss.com/au/software/statistics/>.
- ISO/IEC/IEEE (2010). *Systems and Software Engineering — Vocabulary*. New York.
- iSSEc Project (2009). *Graduate Software Engineering 2009 (GSWE2009): Curriculum Guideline for Graduate Degree Programs in Software Engineering*: IEEE Computer Society/Association for Computing Machinery.
- Jackson, D. (2006). 'Dependable Software by Design.' *Scientific American*, vol. 294, no. 6 (June), pp. 59 – 65.
- Jackson, J. R. (1959). 'Learning From Experience in Business Decision Games.' *California Management Review*, vol. 1, no. 1, pp. 23 – 29.
- Jain, A. and Boehm, B. (2006). 'SimVBSE: Developing a Game for Value-Based Software Engineering.' *Proceedings of the 19th Conference on Software Engineering Education & Training*, (Turtle Bay, Hawaii 19 - 21 April), pp. 103 – 114. Los Alamitos: IEEE Computer Society Press.
- Janis, I. L. (1971). 'Groupthink.' *Psychology Today*, vol. 5, no. 5 (November), pp. 43 – 46, 74 – 76.
- Johns-Boast, L. and Patch, G. (2010). 'A Win-Win Situation: Benefits of Industry-Based Group Projects.' In A. Gardner and L. Jolly (eds.), *Proceedings of the Australasian Association for Engineering Education Conference (AaeE 2010)*, (Sydney, 5 - 8 December), pp. 355 - 360. Sydney: University of Technology Sydney.
- Joint IS2010 Curriculum Task Force (2010). *Curriculum Guideline for Undergraduate Degree Programs in Information Systems*: Association for Computing Machinery and Association for Information Systems.
- Joint Task Force on Computing Curriculum (2004). *Software Engineering 2004: Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering*: IEEE Computer Society/Association for Computing Machinery.
- Joyce, L. (2005). 'Military Apps Drive Simulation Tools.' *R & D*, vol. 47, no. 6, p. 16.
- Kaplan, A. (1973). *The Conduct of Inquiry: Methodology for Behavioral Science*. Aylesbury: Intertext Books
- Keen, P., G. W. (1981). 'Information Systems and Organizational Change.' *Communications of the ACM*, vol. 24, no. 1 (January), pp. 24 – 33.
- Kennedy, J. L. (1971a). 'Simulation Study of Competition in an "Open World".' *Journal of Applied Psychology*, vol. 55, no. 1 (February), pp. 42 – 45.
- Kennedy, J. L. (1971b). 'The System Approach: A Preliminary Exploratory Study of the Relation Between Team Composition and Financial Performance in Business Games.' *Journal of Applied Psychology*, vol. 55, no. 1 (February), pp. 46 – 49.
- Kibbee, J. M., Craft, C. J. and Nanus, B. (1961). *Management Games: A New Technique for Executive Development*. New York: Reinhold Publishing Corporation
- Kirk, J. and Miller, M. L. (1986). *Reliability and Validity in Qualitative Research*. London: Sage Publications

-
- Kitchenham, B. A. (2004). *Procedures for Performing Systematic Reviews*: Keele University, Staffordshire.
- Klabbers, J. H. G., Swart, R. J., van Ulden, A. P. and Vellinga, P. (1994). 'Climate Policy: Management of Organized Complexity Through Gaming.' In D. Crookall and K. Arai (eds.), *Simulation and Gaming Across Disciplines and Cultures*, pp. 122 – 133. Thousand Oaks: Sage Publications.
- Kling, R. and Iacono, S. (1984). 'The Control of Information Systems Developments After Implementation.' *Communications of the ACM*, vol. 27, no. 12 (December), pp. 1218 – 1226.
- Knauss, E., Schneider, K. and Stapel, K. (2008). 'A Game for Taking Requirements Engineering More Seriously.' *Proceedings of the 3rd International Workshop on Multimedia and Enjoyable Requirements Engineering - Beyond Mere Descriptions and with More Fun and Games*, (Barcelona, Spain, 9 September), pp. 22 – 26. Los Alamitos: IEEE Computer Society Press.
- Knotts, U. S. and Keys, J. B. (1997). 'Teaching Strategic Management with a Business Game.' *Simulation & Gaming*, vol. 28, no. 4 (December), pp. 377 – 394.
- Kolb, D. A. (1984). *Experiential Learning: Experience as the Source of Learning and Development*. Englewood Cliffs: Prentice-Hall
- Kozma, R. B. (1994). 'Will Media Influence Learning?' *Educational Technology Research and Development*, vol. 42, no. 2, pp. 7 – 19.
- Kraemer, R. E. and Bessemer, D. W. (1987). *U. S. Tank Platoon Training for the 1987 Canadian Army Trophy (CAT) Competition Using a Simulation Networking (SIMNET) System* (Research Report ARI-RR-1457). Alexandria, Virginia: Army Research Institution for the Behavioral and Social Sciences.
- Kriz, W. C. (2009). 'Bridging the Gap: Transforming Knowledge into Action Through Gaming and Simulation.' *Simulation & Gaming*, vol. 40, no. 1 (February), pp. 28 – 29.
- Kruchten, P. (2005). 'Editor's Introduction: Software Design in a Postmodern Era.' *IEEE Software*, vol. 22, no. 2, pp. 16 – 18.
- Kruchten, P. B. (2004). 'The Nature of Software: What's So Special About Software Engineering?' [on-line]. Available at www.ibm.com/developerworks/rational/library/4700.html. Last accessed 2011-09-30.
- Kulik, C. C. and Kulik, J. A. (1991). 'Effectiveness of Computer-Based Instruction: An Updated Analysis.' *Computers in Human Behavior*, vol. 7, no. 1 – 2, pp. 75 – 94.
- Kulik, J. A., Kulik, C. C. and Bangert-Drowns, R. L. (1985). 'Effectiveness of Computer-Based Education in Elementary Schools.' *Computers in Human Behavior*, vol. 1, pp. 59 – 74.
- Lane, D. C. (1995). 'On a Resurgence of Management Simulation Games.' *Journal of the Operational Research Society*, vol. 46, no. 5 (May), pp. 604 – 625.
- Langley, P. A., Morecroft, J. D. W. and Morecroft, L. E. (1999). *The Oil Producers' Microworld* [computer software], Princes Risborough, UK: Global Strategy Dynamics Ltd. Available from <http://www.strategydynamics.com/>.
- Larman, C. (2006). *Agile and Iterative Development: A Manager's Guide*. Boston: Addison-Wesley

-
- Larman, C. and Basili, V. R. (2003). 'Iterative and Incremental Development: A Brief History.' *IEEE Computer*, vol. 36, no. 6 (June), pp. 47 – 56.
- Lay, M. C., Paku, L. K. and Swan, J. E. (2008). 'Work Placement Reports: Student Perceptions.' In L. Mann, A. Thompson and P. Howard (eds.), *Proceedings of the 19th Annual Conference of the Australasian Association for Engineering Education: To Industry and Beyond*. Barton, ACT: Institution of Engineers, Australia.
- Lee, J. (1999). 'Effectiveness of Computer-Based Instructional Simulation: A Meta Analysis.' *International Journal of Instructional Media*, vol. 26, no. 1, pp. 71– 85.
- Lethbridge, T. C., LeBlanc, R. J., Sobel, A. E. K., Hilburn, T. B. and Diaz-Herrera, J. L. (2006). 'SE2004: Recommendations for Undergraduate Software Engineering Curriculum.' *IEEE Software*, vol. 23, no. 6 (November/December), pp. 19 – 25.
- Levin, B. B., (ed.) (2001). *Energizing Teacher Education and Professional Development with Problem-Based Learning*. Alexandria: Association for Supervision and Curriculum Development.
- Levin, B. B., Dean, C. D. and Pierce, J. W. (2001). 'Frequently Asked Questions About Problem-Based Learning.' In B. B. Levin (ed.) *Energizing Teacher Education and Professional Development with Problem-Based Learning*, pp. 121 – 132. Alexandria: Association for Supervision and Curriculum Development.
- Levin, H. G. and Forman, P. B. (1973). 'A Study of Retention of Knowledge of Neurosciences Information.' *Journal of Medical Education*, vol. 48, no. 9, pp. 867 – 869.
- Levy, D., Kummerfeld, R., Chawla, S., Calvo, R. A. and Fekete, A. (2008). 'The New Software Engineering Program at the University of Sydney.' *Proceedings of the 2008 AaeE Conference*, (Yeppoon). Sydney: Australasian Association for Engineering Education.
- Lewin, K. (1952). *Field Theory in Social Sciences*. London: Tavistock Publications Ltd
- Lincoln, Y. S. and Guba, E. G. (1984). *Naturalistic Inquiry*. London: Sage Publications
- Lomi, A., Larsen, E. R. and Ginsberg, A. (1997). 'Adaptive Learning in Organizations: A System Dynamics-Based Exploration.' *Journal of Management*, vol. 23, no. 4 (July – August), pp. 561 – 583.
- Lundy, J. (1991). 'Cognitive Learning from Games: Student Approaches to Business Games.' *Studies in Higher Education*, vol. 16, no. 2, pp. 179 – 188.
- Lyneis, J. M. and Ford, D. N. (2007). 'System Dynamics Applied to Project Management: A Survey, Assessment, and Directions for Future Research.' *System Dynamics Review*, vol. 23, no. 2 – 3, pp. 157 – 189.
- Macedonia, M. (2002). 'Games Soldiers Play.' *IEEE Spectrum*, vol. 39, no. 3 (March), pp. 32 – 37.
- Macedonia, M. (2005). 'Ender's Game Redux.' *IEEE Computer*, vol. 38, no. 2 (February), pp. 95 – 97.
- Machuca, J. A. D. (2000). 'Transparent-Box Business Simulators: An Aid to Manage the Complexity of Organizations.' *Simulation & Gaming*, vol. 31, no. 2 (June), pp. 230 – 239.

-
- Madachy, R. J. (1996). 'System Dynamics Modeling of an Inspection-Based Process.' *Proceedings of the 18th International Conference on Software Engineering*, (Berlin, Germany), pp. 376 – 386. Los Alamitos: IEEE Computer Society Press.
- Maier, F. H. and Grossler, A. (2000). 'What Are We Talking About? — A Taxonomy of Computer Simulations to Support Learning.' *System Dynamics Review*, vol. 16, no. 2 (Summer), pp. 135 – 148.
- Mandl-Striegnitz, P. (2001). 'How to Successfully Use Software Project Simulation for Educating Software Project Managers.' *Proceedings of the 31st Frontiers in Education Conference*, (Reno, Nevada, 10 - 13 October), pp. T2D-19-24. Los Alamitos: IEEE Computer Society.
- Marshall, C. and Rossman, G. B. (2005). *Designing Qualitative Research*, 4th edition. Thousand Oaks: Sage Publications
- Marshall, M. N. (1996). 'Sampling for Qualitative Research.' *Family Practice*, vol. 13, no. 6, pp. 522 – 525.
- Martin, R. and Raffo, D. M. (2001). 'Application of a Hybrid Process Simulation Model to a Software Development Project.' *The Journal of Systems and Software*, vol. 59, no. 3, pp. 237 – 246.
- Martin, R. C. (2002a). *Agile Software Development: Principles, Patterns, and Practices*. Upper Saddle River: Prentice-Hall
- Martin, R. C. (2009). *Clean Code: A Handbook of Agile Software Craftsmanship*. Upper Saddle River: Prentice Hall
- Martin, R. H. (2002b). *A Hybrid Model of the Software Development Process*. Unpublished PhD dissertation, Portland State University, Portland, Oregon.
- Maruyama, M. (1963). 'The Second Cybernetics: Deviation-Amplifying Mutual Causal Processes.' *American Scientist*, vol. 51, no. 2, pp. 164 – 179.
- Mastaglio, T. W. and Callahan, R. (1995). 'A Large-Scale Complex Virtual Environment for Team Training.' *IEEE Computer*, vol. 28, no. 7 (July), pp. 49 – 56.
- Mathiassen, L., Borum, F. and Pedersen, J. S. (1999). 'Developing Managerial Skills in IT Organizations— A Case Study Based on Action Learning.' *The Journal of Strategic Information Systems*, vol. 8, no. 2, pp. 209 – 225.
- Maxwell, J. A. (2004). *Qualitative Research Design: An Interactive Approach*, 2nd edition. Thousand Oaks: Sage Publications
- Maxwell, N. L., Mergendoller, J. R. and Bellisimo, Y. (2004). 'Developing a Problem-Based Learning Simulation.' *Simulation & Gaming*, vol. 35, no. 4 (December), pp. 488 – 498.
- Mayer, I. S. (2009). 'The Gaming of Policy and the Politics of Gaming: A Review.' *Simulation & Gaming*, vol. 40, no. 6 (December), pp. 825 – 862.
- McBreen, P. (2001). *Software Craftsmanship: The New Imperative*. Upper Saddle River: Addison-Wesley Professional
- McCabe, T. J. (1976). 'A Software Complexity Measure.' *IEEE Transactions on Software Engineering*, vol. 2, no. 4 (December), pp. 308 – 320.
- McCall, J. (2011). *Gaming the Past: Using Video Games to Teach Secondary History* London: Routledge

-
- McConnell, S. (1996). *Rapid Development: Taming Wild Software Schedules*. Redmond: Microsoft Press
- McConnell, S. (2004). *Professional Software Development*. Boston: Addison-Wesley
- McKenna, R. J. (1991). 'Business Computerized Simulation: The Australian Experience.' *Simulation & Gaming*, vol. 22, no. 1 (March), pp. 36 – 62.
- McKenney, J. L. (1962). 'An Evaluation of a Business Game in an MBA Curriculum.' *The Journal of Business*, vol. 35, no. 3 (July), pp. 278 – 286.
- McKenney, J. L. and Dill, W. R. (1966). 'Influences on Learning in Simulation Games.' *American Behavioral Scientist*, vol. 10, no. 3 (October), pp. 28 – 32.
- Meadows, D. H. and Robinson, J. M. (1985). *The Electronic Oracle: Computer Models and Social Decisions*. New York: John Wiley & Sons
- Meadows, D. L. (1989). 'Gaming to Implement System Dynamics Models.' In P. M. Milling and E. O. K. Zahn (eds.), *Computer-Based Management of Complex Systems*, pp. 635 – 640. Berlin: Springer-Verlag.
- Meadows, D. L. (1999). 'Learning to Be Simple: My Odyssey with Games' *Simulation & Gaming*, vol. 30, no. 3, pp. 342 – 351.
- Melody, M. M. (2003). 'A License to Practice Software Engineering.' *IEEE Software*, vol. 20, no. 3 (May/June), pp. 112 – 113.
- Merriam, S. B. (1998). *Qualitative Research and Case Study Applications in Education*. San Francisco: Jossey-Bass
- Merton, R. K. (1948). 'The Self-Fulfilling Prophecy.' *Antioch Review*, vol. 8, pp. 193 - 210.
- Meso, P. and Jain, R. (2006). 'Agile Software Development: Adaptive Systems Principles and Best Practices.' *Information Systems Management*, vol. 23, no. 3 (June), pp. 19 – 30.
- Michael, D. and Chen, S. (2005). *Serious Games: Games That Educate, Train, and Inform*. Boston: Thomson Course Technology PTR
- Miles, M. B. and Huberman, A. M. (1994). *Qualitative Data Analysis: An Expanded Sourcebook*, 2nd ed. Thousand Oaks: Sage Publications
- Miller, G. A. (1956). 'The Magical Number Seven, Plus or Minus Two: Some Limits on Our Capacity for Processing Information.' *Psychological Review*, vol. 63, no. 2, pp. 81 – 97.
- Miller, J. G. (1978). *Living Systems*. New York: McGraw-Hill Book Company
- Milne, A. A. (1998). *Winnie-the-Pooh*. London: Methuen Children's Books
- Moore, L. F. (1967). 'Business Games versus Cases as Tools of Learning.' *Training and Development Journal*, vol. 21, no. 10 (October), pp. 13 – 23.
- Morecroft, J. D. W. (1988). 'System Dynamics and Microworlds for Policymakers.' *European Journal of Operational Research*, vol. 35, pp. 301 – 320.
- Moskowitz, H. (1973). 'An Experimental Investigation of Decision-Making in a Simulated Research and Development Environment.' *Management Science*, vol. 19, no. 2, pp. 676 – 687.

-
- Murray, H. J. R. (1913). *A History of Chess*. Oxford: Clarendon Press
- Myers, M. D. (1994). 'A Disaster for Everyone to See: An Interpretive Analysis of a Failed IS Project.' *Accounting, Management, and Information Technologies*, vol. 4, no. 4, pp. 185 – 201.
- Naur, P. and Randell, B., (eds.). (1969). *Software Engineering: Report on a Conference Sponsored by the NATO Science Committee, Garmisch, Germany, 7th to 11th October 1968*. Brussels: Scientific Affairs Division, NATO.
- Navarro, E. O. (2006). *SimSE: A Software Engineering Simulation Environment for Software Process Education*. Unpublished PhD thesis, University of California, Irvine,
- Navarro, E. O., Baker, A. and van der Hoek, A. (2004). 'Teaching Software Engineering Using Simulation Games.' *Proceedings of the 2004 International Conference on Simulation in Education*, (San Diego, California, 18 - 21 January). New York: ACM.
- Navarro, E. O. and van der Hoek, A. (2005). 'Design and Evaluation of an Educational Software Process Simulation Environment and Associated Model.' *Proceedings of the 18th Conference on Software Engineering Education and Training*, (Ottawa, Canada 18 - 20 April). Los Alimitos: IEEE Computer Society Press.
- Navarro, E. O. and van der Hoek, A. (2007). 'Comprehensive Evaluation of an Educational Software Engineering Simulation Environment.' *Proceedings of the 20th Conference on Software Engineering Education and Training*, (Dublin, Ireland, 3 - 5 July), pp. 195 – 202. New York: ACM.
- Navarro, E. O. and van der Hoek, A. (2008). 'On the Role of Learning Theories in Furthering Software Engineering Education.' In H. J. C. Ellis, S. A. Demurjian and J. F. Naveda (eds.), *Software Engineering: Effective Teaching and Learning Approaches and Practices*, pp. 38 – 59. Hershey: IGI Global.
- Navarro, E. O. and van der Hoek, A. (2009). 'Multi-Site Evaluation of SimSE.' *Proceedings of the 40th ACM Technical Symposium on Computer Science Education* (Chattanooga, Tennessee, 3 - 7 March). New York: ACM.
- Naveda, J. F. and Seidman, S. B. (2005). 'Professional Certification of Software Engineers: The CSDP Program.' *IEEE Software*, vol. 22, no. 5, pp. 73 – 77.
- Naveda, J. F. and Seidman, S. B., (eds.). (2006). *IEEE Computer Society Real-World Software Engineering Problems: A Self-Study Guide for Today's Software Professional*. Hoboken: John Wiley & Sons.
- Naylor, T. H. and Gattis, D. R. (1976). 'Corporate Planning Models.' *California Management Review*, vol. 18, no. 4 (Summer), pp. 69 – 78.
- Neuhauser, J. J. (1976). 'Business Games Have Failed.' *Academy of Management Review*, vol. 1, no. 4 (October), pp. 124 – 129.
- Neumann, P. G. (1995). *Computer-Related Risks*. Reading, Massachusetts: Addison-Wesley Publishing Company
- Norman, D. A. (1988). *The Psychology of Everyday Things*. New York: Basic Books
- Oh, E. and van der Hoek, A. (2002). 'Towards Game-Based Simulation as a Method of Teaching Software Engineering.' *Proceedings of the 32nd Annual Frontiers in Education Conference*, (6 - 9 November), pp. S2G-13. Los Alimitos: IEEE Computer Society Press.

-
- Oliver, P. (2003). *The Student's Guide to Research Ethics*. Maidenhead: Open University Press
- Osterweil, L. (1987). 'Software Processes are Software Too.' *Proceedings of the 9th International Conference on Software Engineering*, (Monterey, California), pp. 2 – 13. Los Alamitos: IEEE Computer Society Press.
- Papert, S. (1980). *Mindstorms*. Brighton, Sussex: The Harvester Press
- Parasuraman, A. (1981). 'Assessing the Worth of Business Simulation Games: Problems and Prospects.' *Simulation & Games*, vol. 12, no. 2 (June), pp. 189 – 200.
- Patton, M. Q. (2002). *Qualitative Research and Evaluation Methods*, 3rd edition. Thousand Oaks: Sage Publications
- Perla, P. P. (1990). *The Art of Wargaming: A Guide for Professionals and Hobbyists*. Annapolis, Maryland: Naval Institute Press
- Petranek, C. F. (1994). 'A Maturation in Experiential Learning: Principles of Simulation and Gaming.' *Simulation & Gaming*, vol. 25, no. 4 (December), pp. 513 – 522.
- Petranek, C. F., Corey, S. and Black, R. (1992). 'Three Levels of Learning in Simulations: Participating, Debriefing, and Journal Writing.' *Simulation & Gaming*, vol. 23, no. 2 (June), pp. 174 – 185.
- Pfahl, D., Koval, N. and Ruhe, G. (2001). 'An Experiment for Evaluating the Effectiveness of Using a System Dynamics Simulation Model in Software Project Management Education.' *Proceedings of the 7th International Software Metrics Symposium*, (6 - 8 April), pp. 97-109. Los Alimitos: IEEE Computer Society Press.
- Pfahl, D., Laitenberger, O., Dorsch, J. and Ruhe, G. (2003). 'An Externally Replicated Experiment for Evaluating the Learning Effectiveness of Using Simulations in Software Project Management Education.' *Empirical Software Engineering*, vol. 8, no. 4 (December), pp. 367 – 395.
- Pfahl, D., Laitenberger, O., Ruhe, G., Dorsch, J. and Krivobokova, T. (2004). 'Evaluating the Learning Effectiveness of Using Simulations in Software Project Management Education: Results from a Twice Replicated Experiment.' *Information and Software Technology*, vol. 46, no. 2, pp. 127 – 147.
- Pierfy, D. A. (1977). 'Comparative Simulation Game Research: Stumbling Blocks and Steppingstones.' *Simulation & Games*, vol. 8, no. 2 (June), pp. 255 – 268.
- Polack-Wahl, J. A. (2006). 'Lessons Learned From Different Types of Projects in Software Engineering.' *Proceedings of the International Conference on Frontiers in Education: Computer Science & Computer Engineering*, (Las Vegas, Nevada, 26 - 29 June), pp. 258 – 263. New York: CSREA Press.
- Prakash, E., Brindle, G., Jones, K., Zhou, S., Chaudhari, N. S. and Wong, K.-W. (2009). 'Advances in Games Technology: Software, Models, and Intelligence.' *Simulation & Gaming*, vol. 40, no. 6 (December 1, 2009), pp. 752-801.
- Prensky, M. (2006). *Don't Bother Me Mom– I'm Learning!*. St. Paul, Minnesota: Paragon House Publishers
- Prensky, M. (2007). *Digital Game-Based Learning*. St. Paul, Minnesota: Paragon House Publishers
- Prohaska, C. R. and Frank, E. J. (1990). 'Using Simulations to Investigate Management Decision Making.' *Simulation & Gaming*, vol. 21, no. 1 (March), pp. 48 – 58.

-
- Qiu, M. and Chen, L. (2010). 'A Problem-Based Learning Approach to Teaching an Advanced Software Engineering Course.' *Proceedings of the 2nd International Workshop on Education Technology and Computer Science*, (Wuhan, China 6 - 7 March), pp. 252 – 255. Los Alamitos: IEEE Computer Society Press.
- QSR International (2010). *NVivo* [computer software], version 8.0. Available from <http://www.qsrinternational.com>.
- Raia, A. P. (1966). 'A Study of the Educational Value of Management Games.' *The Journal of Business*, vol. 39, no. 3 (July), pp. 339 – 352.
- Randel, J. M., Morris, B. A., Wetzel, C. D. and Whitehill, B. V. (1992). 'The Effectiveness of Games for Educational Purposes: A Review of Recent Research.' *Simulation & Gaming*, vol. 23, no. 3 (September), pp. 261-276.
- Raser, J. R. (1969). *Simulation and Society: An Exploration of Scientific Gaming*. Boston: Allyn and Bacon Inc
- Remus, H. and Zilles, S. (1979). 'Prediction and Management of Program Quality.' *Proceedings of the 4th International Conference on Software Engineering*, (Munich, Germany), pp. 341 – 350. New York: ACM Press.
- Remus, W. (1977). 'Who Likes Business Games?' *Simulation & Games*, vol. 8, no. 4 (December), pp. 469 – 480.
- Remus, W. (1986). 'Graduate Students as Surrogates for Managers in Experiments on Business Decision Making.' *Journal of Business Research*, vol. 14, no. 1 (February), pp. 19 – 25.
- Remus, W. and Jenner, S. (1981). 'Playing Business Games: Expectations and Realities.' *Simulation & Games*, vol. 12, no. 4 (December), pp. 480 – 488.
- Ribaud, V. and Saliou, P. (2008). 'Evolution of an Integrated Course Towards a Sandwich Course.' In A. Cortesi and F. Luccio (eds.), *Proceedings of the 2008 Informatics Education Europe III Conference*, (Venice, Italy, 4 - 5 December), pp. 92 – 104. New York: ACM.
- Richards, L. (2009). *Handling Qualitative Data*, 2nd edition. Thousand Oaks: Sage Publications
- Richardson, G. P. (1986). 'Problems with Causal-Loop Diagrams.' *System Dynamics Review*, vol. 2, no. 2 (Summer), pp. 158 – 170.
- Richardson, G. P. (1991). *Feedback Thought in Social Science and Systems Theory*. Waltham: Pegasus Communications
- Richardson, G. P. (1997). 'Problems With Causal-Loop Diagrams Revisited.' *System Dynamics Review*, vol. 13, no. 3 (Fall), pp. 247 – 252.
- Riddell, R. (1997). 'Doom Goes to War.' *Wired*, vol. 5, no. 4 (April), pp. 113 – 118, 164 – 166.
- Riehle, R. D. (2008). *An Engineering Context for Software Engineering*. Unpublished PhD thesis, Naval Postgraduate School, Monterey, California.
- Ritter, G. (1958). *The Schlieffen Plan: Critique of a Myth*. Translated by A. Wilson and E. Wilson. London: Oswald Wolff (Publishers) Ltd
- Roberts, E. B. (1964). *The Dynamics of Research and Development*. New York: Harper & Row

-
- Roberts, N., Andersen, D. F., Deal, R. M., Garet, M. S. and Shaffer, W. A. (1983). *Introduction to Computer Simulation: A System Dynamics Modeling Approach*. Portland: Productivity Press
- Robillard, P. N. and Robillard, M. (1998). 'Improving Academic Software Engineering Projects: A Comparative Study of Academic and Industry Projects.' *Annals of Software Engineering*, vol. 6, no. 1, pp. 343 – 363.
- Rodriguez, D., Sicilia, M. A., Cuadrado-Gallego, J. J. and Pfahl, D. (2006). 'e-Learning in Project Management Using Simulation Models: A Case Study Based on the Replication of an Experiment.' *IEEE Transactions on Education*, vol. 49, no. 4, pp. 451– 463.
- Ronis, D. L. (2008). *Problem-Based Learning for Math & Science : Integrating Inquiry and the Internet*, 2nd edition. Thousand Oaks: Corwin Press
- Rossman, G. B. and Rallis, S. F. (1998). *Learning in the Field: An Introduction to Qualitative Research*. Thousand Oaks: Sage Publications
- Rousseau, D. and Hayes-Roth, B. (1996). *Personality in Synthetic Agents* (KSL 96-21). Stanford: Department of Computer Science, Stanford University.
- Royce, W. (1998). *Software Project Management: A Unified Framework*. Reading, Massachusetts: Addison-Wesley
- Royce, W. W. (1970). 'Managing the Development of Large Software Systems.' *Proceedings of IEEE Wescon*, pp. 1 – 9. Los Alamitos: IEEE Computer Society Press.
- Rusu, A., Russell, R., Robinson, J. and Rusu, A. (2010). 'Learning Software Engineering Basic Concepts Using a Five-Phase Game.' *Proceedings of the 40th ASEE/IEEE Frontiers in Education Conference*, (Washington, DC, 27 - 30 October), pp. 2SD-1 – 2SD1-6. Washington: IEEE Computer Society.
- Ryan, T. (2000). 'The Role of Simulation Gaming in Policy-Making.' *Systems Research and Behavioral Science*, vol. 17, no. 4 (July – August), pp. 359 – 364.
- Sauer, C. (1993). *Why Information Systems Fail: A Case Study Approach*. Henley-on-Thames: Alfred Waller Limited
- Savery, J. R. and Duffy, T. M. (1994). 'Problem Based Learning: An Instructional Model and its Constructivist Framework.' *Educational Technology*, vol. 35, no. 5, pp. 31 – 38.
- Savin-Baden, M. (2003). *Facilitating Problem-Based Learning*. Maidenhead: The Society for Research into Higher Learning & Open University Press
- Savin-Baden, M. and Major, C. H. (2004). *Foundations of Problem-Based Learning*. Maidenhead: The Society for Research into Higher Learning & Open University Press
- Savin-Baden, M. and Wilkie, K. (1996). *Problem-Based Learning Online*. Maidenhead, Berkshire: Open University Press
- Sayre, F. (1911). *Map Maneuvers and Tactical Rides*, 4th edition. Springfield, Massachusetts: Springfield Printing & Binding Company
- Schaffer, W. M. (1984). 'Stretching and Folding in Lynx Fur Returns: Evidence for a Strange Attractor in Nature?' *American Naturalist*, vol. 124, no. 6, pp. 798 – 820.

-
- Schein, E. H. (1980). *Organizational Psychology*, 3rd edition. Englewood Cliffs: Prentice-Hall
- Schlimmer, J. C., Fletcher, J. B. and Hermens, L. A. (1994). 'Team-Oriented Software Practicum.' *IEEE Transactions on Education*, vol. 37, no. 2 (May), pp. 212 – 220.
- Schott, B. (1976). 'Using a Business Game to Teach Risk Management.' *Journal of Risk and Insurance*, vol. 43, no. 3 (September), pp. 526 – 532.
- Schrage, M. and Peters, T. (1999). *Serious Play : How the World's Best Companies Simulate to Innovate*. Boston: Harvard Business School Press
- Schumann, P. L., Anderson, P. H. and Scott, T. W. (1996). 'Introducing Ethical Dilemmas into Computer-Based Simulation Exercises to Teach Business Ethics.' *Developments in Business Simulations and Experiential Exercises*, vol. 23, pp. 74 – 80.
- Schwaber, K. (2004). *Agile Project Management with Scrum*. Redmond: Microsoft Press
- Schwartz, P., Mennin, S. and Webb, G. (2001). *Problem-Based Learning: Case Studies, Experience and Practice*. New York: Routledge
- Sebern, M. J. (2002). 'The Software Development Laboratory: Incorporating Industrial Practice in an Academic Environment.' *Proceedings of the 15th Conference on Software Engineering Education and Training*, (Covington, Kentucky, 25 - 27 February), pp. 118 – 123. Los Alamitos: IEEE Computer Society.
- Selye, H. (1974). *Stress Without Distress*. Philadelphia: Signet Books
- Selye, H. (1978). *The Stress of Life*, 2nd edition. New York: McGraw-Hill
- Senge, P. M. (1990). *The Fifth Discipline: The Art & Practice of The Learning Organization*. Milsons Point: Random House
- Senge, P. M. (2006). *The Fifth Discipline: The Art & Practice of The Learning Organization*, Revised edition. London: Random House Business Books
- Senge, P. M. and Fulmer, R. M. (1993). 'Simulations, Systems Thinking and Anticipatory Learning.' *Journal of Management Development*, vol. 12, no. 6, pp. 21 – 33.
- Senge, P. M., Kleiner, A., Roberts, C., Ross, R. B. and Smith, B. J. (1994). *The Fifth Discipline Fieldbook*. London: Nicholas Brealey Publishing
- Sengupta, K., Abdel-Hamid, T. K. and Bosley, M. (1999). 'Coping with Staffing Delays in Software Project Management: An Experimental Investigation.' *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, vol. 29, no. 1, pp. 77 – 91.
- Sharp, H. and Hall, P. (2000). 'An Interactive Multimedia Software House Simulation for Postgraduate Software Engineers.' *Proceedings of the 22nd International Conference on Software Engineering*, (Limerick, Ireland), pp. 688 – 691. New York: ACM.
- Shaw, K. and Dermoudy, J. (2005). 'Engendering an Empathy for Software Engineering.' *Proceedings of the 7th Australasian Conference on Computing Education*, (Newcastle, New South Wales), pp. 135 – 144. Sydney: Australian Computer Society.
- Shim, J. K. (1978). 'Management Game Simulation: Survey and New Direction.' *University of Michigan Business Review*, vol. 30, no. 3 (May), pp. 26 – 29.

-
- Simon, H. A. (1957). *Models of Man Social and Rational: Mathematical Essays on Rational Human Behavior in a Social Setting*. New York: John Wiley & Sons
- Simon, H. A. (1996). *The Sciences of the Artificial*, 3rd edition. Cambridge, Massachusetts: The MIT Press
- Simons, K. L. (1993). 'New Technologies in Simulation Games.' *System Dynamics Review*, vol. 9, no. 2 (Summer), pp. 135 – 152.
- Singer, P. W. (2010). 'Meet the Sims... and Shoot Them. The Rise of Militainment.' *Foreign Policy*, no. 178 (March/April), pp. 91 – 95.
- Sitzmann, T. (2011). 'A Meta-Analytic Examination of the Instructional Effectiveness of Computer-Based Simulation Games.' *Personnel Psychology*, vol. 64, no. 2, pp. 489 – 528.
- Smith, R. (2010). 'The Long History of Gaming in Military Training.' *Simulation & Gaming*, vol. 41, no. 1 (February), pp. 6-19.
- Smith, R. D. (1998). 'Essential Techniques for Military Modeling & Simulation.' In D. J. Medeiros, E. F. Watson, J. S. Carson and M. S. Manivannan (eds.), *Proceedings of the 1998 Winter Simulation Conference*, (Washington, D.C.), pp. 805 – 812. Los Alamitos: IEEE Computer Society Press.
- Sobkowiak, R. T. and LeBleu, R. E. (1994). 'A Proving Ground for Organizational Change.' *Computerworld*, vol. 28, no. 41 (10 October), p. 41.
- Sommerville, I. (2007). *Software Engineering*, 8th edition. Addison-Wesley
- Starr, P. (1984). *The Social Transformation of American Medicine: The Rise of a Sovereign Profession and the Making of a Vast Industry*. New York: Basic Books
- Sterling, B. (1993). 'War is Virtual Hell.' *Wired*, vol. 1, no. 1 (March/April), pp. 46 - 51, 94 - 99.
- Sterman, J. D. (1988). *People Express Management Flight Simulator* [computer software], Banbury, UK: Phrontis Limited.
- Sterman, J. D. (1989). 'Modeling Managerial Behavior: Misperceptions of Feedback in a Dynamic Decision Making Environment.' *Management Science*, vol. 35, no. 3 (March), pp. 321 – 339.
- Sterman, J. D. (1994). 'Learning In And About Complex Systems.' *System Dynamics Review*, vol. 10, no. 2 – 3 (Summer - Fall), pp. 291 – 330.
- Sterman, J. D. (2000). *Business Dynamics: Systems Thinking and Modelling for a Complex World*. New York: Irwin McGraw-Hill
- Stix, G. (1994). 'Aging Airways.' *Scientific American*, vol. 270, no. 5 (May), pp. 96 –104.
- Strauss, A. L. (1987). *Qualitative Analysis for Social Scientists*. Cambridge: Cambridge University Press
- Suits, B. (1967). 'What is a Game?' *Philosophy of Science*, vol. 34, no. 2 (June), pp. 148 – 156.
- Suits, B. (2005). 'Construction of a Definition.' In K. Salen and E. Zimmerman (eds.), *The Game Design Reader*, pp. 172 – 191. Cambridge, Massachusetts: The MIT Press.

-
- Sutherland, J. and van den Heuvel, W.-J. (2002). 'Enterprise Application Integration and Complex Adaptive Systems.' *Communications of the ACM*, vol. 45, no. 10 (October), pp. 59 – 64.
- Tan, J., Wen, H. J. and Awad, N. (2005). 'Health Care and Services Delivery Systems as Complex Adaptive Systems.' *Communications of the ACM*, vol. 48, no. 5 (May), pp. 36 – 44.
- Tang, C., Lai, P., Tang, W., Davis, H., Frankland, S., Oldfield, K., Walters, M., Ng, M. L., Tse, P., Taylor, G., Tiwari, A., Yim, M. and Yuen, E. (1997). 'Developing a Context-Based PBL Model.' In J. Conway, R. Fisher, L. Sheridan-Burns and G. Ryan (eds.), *Research and Development in Problem Based Learning: Integrity, Innovation, Integration*, pp. 588 – 589. Newcastle: Australian Problem Based Learning Network.
- Taran, G. (2007). 'Using Games in Software Engineering Education to Teach Risk Management.' *Proceedings of the 20th Conference on Software Engineering Education & Training*, (Dublin, Ireland, 3 - 5 July), pp. 211 – 220. Los Alimitos: IEEE Computer Society Press.
- Tennyson, R. D. (1994). 'The Big Wrench vs. Integrated Approaches: The Great Media Debate.' *Educational Technology Research and Development*, vol. 42, no. 3, pp. 15 – 28.
- Thomas, C. J. and Deemer, W. L. (1957). 'The Role of Operational Gaming in Operations Research.' *Operations Research*, vol. 5, no. 1 (February), pp. 1 – 27.
- Tipton, R. (2000). 'Make 5/1/00 National "Hug-a-Programmer" Day.' *NEWS/400*, no. 3 (March), p. 17.
- Toffler, A. (1970). *Future Shock*. London: The Bodley Head
- Toth, F. L. (1994). 'Policy Exercises: The First 10 Years.' In D. Crookall and K. Arai (eds.), *Simulation and Gaming Across Disciplines and Cultures*, pp. 257 – 264. Thousand Oaks: Sage Publications.
- Truex, D. P., Baskerville, R. and Klein, H. (1999). 'Growing Systems in Emergent Organizations.' *Communications of the ACM*, vol. 42, no. 8 (August), pp. 117 – 123.
- Tvedt, J. D. (1996). *An Extensible Model for Evaluating the Impact of Process Improvements on Software Development Cycle Time*. Unpublished PhD thesis, Arizona State University, Phoenix, Arizona.
- van der Meij, H., Albers, E. and Leemkuil, H. (2011). 'Learning from Games: Does Collaboration Help?' *British Journal of Educational Technology*, vol. 42, no. 4, pp. 655 – 664.
- Van Joolingen, W. R. and De Jong, T. (1998). 'Scientific Discovery Learning with Computer Simulations of Conceptual Domains.' *Review of Educational Research*, vol. 68, no. 2, pp. 179 – 201.
- van Vliet, H. (2006). 'Reflections on Software Engineering Education.' *IEEE Software*, vol. 23, no. 3 (May/June), pp. 55 – 61.
- VanSickle, R. L. (1986). 'A Quantitative Review of Research on Instructional Gaming: A Twenty-Year Perspective.' *Theory and Research in Social Education*, vol. 14, no. 3, pp. 245 – 264.

-
- Variante, T., Rosetta, B., Steffen, M., Rubin, H. and Yourdon, E. (1994). 'Modeling the Maintenance Process.' *American Programmer*, vol. 7, no. 3 (March), pp. 29 – 37.
- Veglahn, P. A., Frazer, J. R. and Bommer, M. R. W. (1978). 'Computer Simulation—Training Tool for Collective Bargaining.' *Personnel Journal*, vol. 57, no. 11 (November), pp. 614 – 617.
- Visiongain (2010). *The Military Simulation and Virtual Training Market 2010-2020*. London: Visiongain.
- Vogel, J. J., Vogel, D. S., Cannon-Bowers, J., Bowers, C. A., Muse, K. and Wright, M. (2006). 'Computer Gaming and Interactive Simulations for Learning: A Meta-Analysis.' *Journal of Educational Computing Research*, vol. 34, no. 3, pp. 229-243.
- von Bertalanffy, L. (1968). *General System Theory*. New York: George Braziller
- Wang, A. I., Øfsdahl, T. and Morch-Storstein, O. K. (2008). 'An Evaluation of a Mobile Game Concept for Lectures.' *Proceedings of the 21 Conference on Software Engineering Education and Training*, (Charleston, South Carolina, 14 - 17 April), pp. 197 – 204. Los Alimitos: IEEE Computer Society Press.
- Wang, T. and Zhu, Q. (2009). 'A Software Engineering Education Game in a 3-D Online Virtual Environment.' *Proceedings of the 1st International Workshop on Education Technology and Computer Science*, (Wuhan, Hubei, China, 7 - 8 March, 2009), pp. 708 – 710. Los Alimitos: IEEE Computer Society Press.
- Watson, H. J. and Blackstone, J. H. (1989). *Computer Simulation*, 2nd edition. New York: John Wiley & Sons
- Watt, K. E. F. (1977). 'Why Won't Anyone Believe Us?' *Simulation*, vol. 28, no. 1 (January), pp. 1 – 3.
- Weinberg, G. M. (1998). *The Psychology of Computer Programming*, silver anniversary edition. New York: Dorset Housing Publishing
- Wenzler, I. (2009). 'The Ten Commandments for Translating Simulation Results into Real-Life Performance.' *Simulation & Gaming*, vol. 40, no. 1 (February), pp. 98 – 109.
- Williams, D. (1978). 'Computer Games that Planners Play.' *Business Week*, no. 2565 (18 December), p. 66.
- Wilson, A. (1968). *The Bomb and the Computer*. London: Barrie & Rockliff, The Cresset Press
- Wilson, K. A., Bedwell, W. L., Lazzara, E. H., Salas, E., Burke, C. S., Estock, J. L., Orvis, K. L. and Conkey, C. (2009). 'Relationships Between Game Attributes and Learning Outcomes.' *Simulation & Gaming*, vol. 40, no. 2 (April), pp. 217 – 266.
- Wolfe, J. (1978). 'The Effects of Game Complexity on the Acquisition of Business Policy Knowledge.' *Decision Sciences*, vol. 9, no. 1 (January), pp. 143 – 155.
- Wolfe, J. (1997). 'The Effectiveness of Business Games in Strategic Management Course Work.' *Simulation & Gaming*, vol. 28, no. 4 (December), pp. 360 – 376.
- Wolfe, J. and Fritzsche, D. J. (1998). 'Teaching Business Ethics with Management and Marketing Games.' *Simulation & Gaming*, vol. 29, no. 1 (March), pp. 44 – 59.

-
- Wolfe, J. and Guth, G. R. (1975). 'The Case Approach versus Gaming in the Teaching of Business Policy: An Experimental Evaluation.' *The Journal of Business*, vol. 48, no. 3 (July), pp. 349 – 364.
- Wolstenholme, E. F. (1990). *System Enquiry: A System Dynamics Approach*. Brisbane: John Wiley & Sons
- Woods, D. R. (1996). 'Problem-Based Learning for Large Classes in Chemical Engineering.' In L. Wilkserson and W. H. Gijsselaers (eds.), *Bringing Problem-Based Learning to Higher Education: Theory and Practice*, pp. 91 – 99. New York: Jossey-Bass.
- Woolfolk, A. E. (2009). *Educational Psychology*, 11th edition. Upper Saddle River: Prentice Hall
- Ye, E., Chang, L. and Polack-Wahl, J. A. (2007). 'Enhancing Software Engineering Education Using Teaching Aids in 3-D Online Virtual Worlds.' *Proceedings of the 37th Annual Frontiers In Education Conference - Global Engineering: Knowledge Without Borders, Opportunities Without Passports*, (San Antonio, Texas, 18 - 21 October), pp. T1E-8 – T1E-13. Los Alamitos: IEEE Computer Society Press.
- Yeh, R. T. (2002). 'Educating Future Software Engineers.' *IEEE Transactions on Education*, vol. 45, no. 1 (February), pp. 2 – 3.
- Yourdon, E. (1992). *Decline and Fall of the American Programmer*. Sydney: Prentice-Hall
- Yourdon, E. (1998). *Rise and Resurrection of the American Programmer*. Sydney: Prentice-Hall
- Yourdon, E. (2000). 'Y2K Success Lessons.' *Computerworld*, vol. 34, no. 4 (24 January), p. 40.
- Yourdon, E. (2004). *Death March*, 2nd edition. Upper Saddle River: Prentice Hall
- Zagal, J. P., Rick, J. and Hsi, I. (2006). 'Collaborative Games: Lessons Learned from Board Games.' *Simulation & Gaming*, vol. 37, no. 1 (March), p. 24 — 40.
- Zapata, C. M. (2010). 'A Classroom Game for Teaching Management of Software Companies.' *Dyna*, vol. 77, no. 163, pp. 290 – 299.
- Zapata, C. M. and Awad-Aubad, G. (2007). 'Requirements Game: Teaching Software Project Management.' *CLEI Electronic Journal*, vol. 10, no. 1 (June).
- Zhu, Q., Wang, T. and Tan, S. (2007). 'Adapting Game Technology to Support Software Engineering Process Teaching: From SimSE to MO-SEProcess.' *Proceedings of the 3rd International Conference on Natural Computation*, (Hainan, China, 25 - 27 August). Washington: IEEE Computer Society.
- Zorpette, G. (1991). 'Emulating the Battlefield.' *IEEE Spectrum*, vol. 28, no. 9 (September), pp. 36 – 39.

Appendices

Appendix A: Causal Loop Diagrams

A causal-loop diagram is a set of variables connected by arrows that denote the causal influences between the variables, and which can be used to elicit, capture, and communicate mental models (Sterman, 2000, p. 137). For instance, consider the following causal-loop diagram:

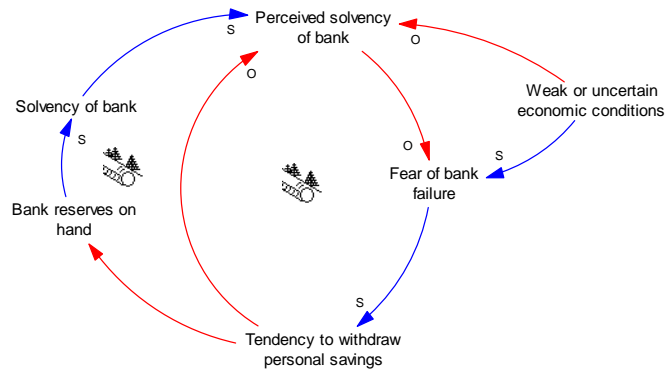


Figure 14: A simple causal loop diagram

This causal-loop diagram describes a self-fulfilling prophecy, a false definition of a situation that evokes a new behaviour which makes the originally false conception come true (Merton, 1948, p. 195; Richardson, 1991, pp. 83 - 84). In this case, Depression-era bank depositors, fearing for the solvency of their bank for some reason (rumour, speculation, national economic performance...), decide to withdraw their funds. The solvency of the bank is slightly affected but, more so, less skittish depositors see this happening and decide it is now time to withdraw their funds, and so the snowball grows until the solvency of the bank is truly affected.

The arrows in the diagram represent a causal link or relationship between two variables, between the cause and the effect, meaning that when one changes it will affect the other. Such a relationship is causal “if it is necessarily sequential in time and incorporates some hypothesis about the mechanisms whereby one element directly influences another” (Meadows & Robinson, 1985, p. 11). The S (sometimes represented as a +) and the O (sometimes represented as a -) at the head of the arrows indicate the polarity, or the way in which this change will happen (Maruyama, 1963, pp. 175 - 176):

S, or +, means that if the cause changes, the effect will change in the same direction (either increase or decrease) beyond what it would have otherwise done. O, or -, means that if the cause changes, the effect will change in the opposite direction (either increase or decrease) beyond what it would have otherwise done.

The key phrase in the above descriptions is *beyond what it would have otherwise done*. A change in a cause variable may not necessarily mean the effect will also change because there may be other factors feeding into the effect. Therefore, to know what actually happens, we need to also know how all the other variables are changing at that time (Richardson, 1986, p. 161; Sterman, 2000, p. 139). In this quest, there is some debate over whether the S/O or +/- notation is better, or whether an altogether fresh notation is needed (Richardson, 1997; Ford, 1999, p. 82). Still, if a causal-loop diagram is being used as an initial exploration of a topic, and changes in other parts of the diagram are appreciated, then the choice of either notation should be a personal preference.

Where a series of causal relationships form a loop, for example, *Perceived solvency of bank* → *Tendency to withdraw personal savings* → *Perceived solvency of bank*, then the loop itself has a polarity (Roberts et al., 1983, p. 40):

When the number of opposite causal relationships (those with O or – at the head) in the loop is an odd number, then the loop represents balancing feedback: changes within the loop feed back to negate or stabilise the original change. Balancing feedback usually seeks a goal, or seeks to reduce the difference between where a system is now and where it should be. As long as there is a difference between the present state of the system and its desired state, balancing feedback will move the system in the direction of the desired change.

When the number of opposite causal relations is even, as in this case, then the loop represents reinforcing feedback: changes within the system amplify the original change in the same direction. This may lead to either growth or decline depending on the starting conditions.

A causal-loop diagram consisting of even three or four such loops can rightly be called complex (Forrester, 1969, p. 108) since the interplay of different behaviours can be difficult to infer. Therefore, any causal-loop diagram needs to be read

critically, particularly to confirm that it depicts causal relations between variables and not just correlations. Correlations among variables reflect the past behaviour of the system, but may not hold in all circumstances. Sterman (2000, pp. 142 - 143) describes a trivial example in which there appears to be a causal relationship between ice-cream sales and a city's murder rate: data points to an increasing murder rate during a time when ice-cream sales also increase. Naturally, drawing a direct causal link between the two is dangerous since it suggests that cutting ice-cream consumption would cut the murder rate. In fact, temperature is more likely to be a causal link affecting both variables.

Causal loop diagrams are a communication tool, not a simulation tool. They help us think about the structure of the system (Ford, 1999, p. 82).

Appendix B: Stock and Flow Diagrams

Whereas causal-loop diagrams emphasise the feedback structure of systems, stock-and-flow diagrams emphasise the underlying physical structure of the system and depict its state at certain points in time (Sterman, 2000, p. 102). For instance, consider the following stock-and-flow diagram:

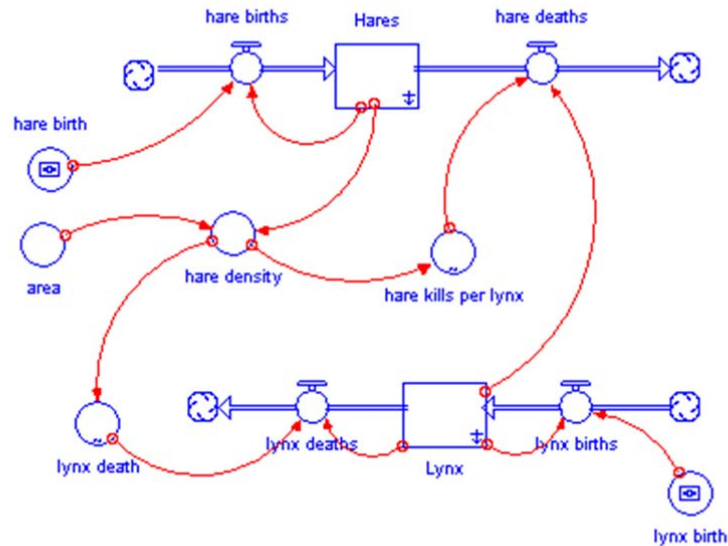


Figure 15: A stock and flow diagram of the classic predator-prey relationship

This stock-and-flow diagram describes a classic predator-prey relationship between the snowshoe hare and lynxes in the area surrounding Hudson Bay in North America in the nineteenth century (Roberts et al., 1983, pp. 119 - 130; Schaffer, 1984) and is based on detailed time-series data of pelts sold to the Hudson Bay Company. The number of hares is a function of the forage they have access to in the area, less the number taken by lynxes. Meanwhile, the number of lynxes is a function of how many hares they can catch, assuming that their birth rate is moderated by current environmental conditions. If the hare population increases to a level that cannot be sustained by the available forage, then their numbers will fall through starvation as well as through predation. The lynxes will thereafter have less prey and their population will be similarly affected. The conditions are then set for a fresh oscillation of both predator and prey as shown in Figure 16.

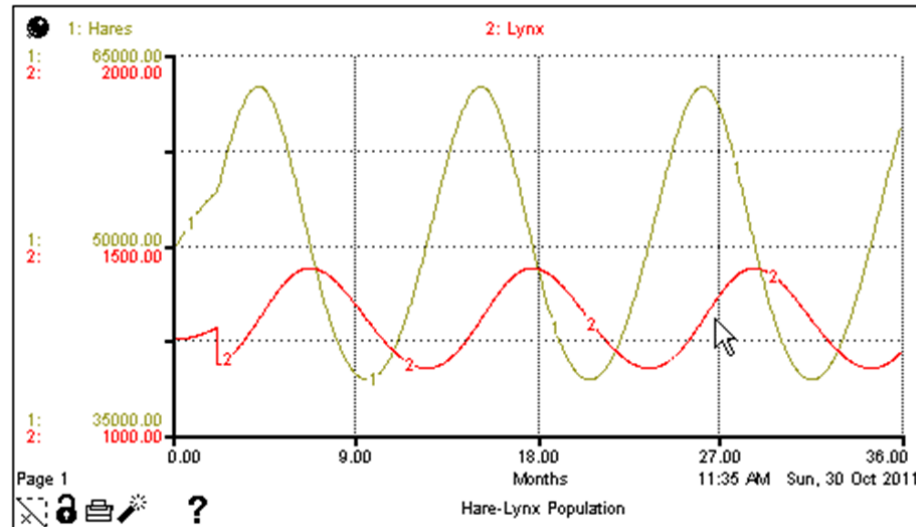


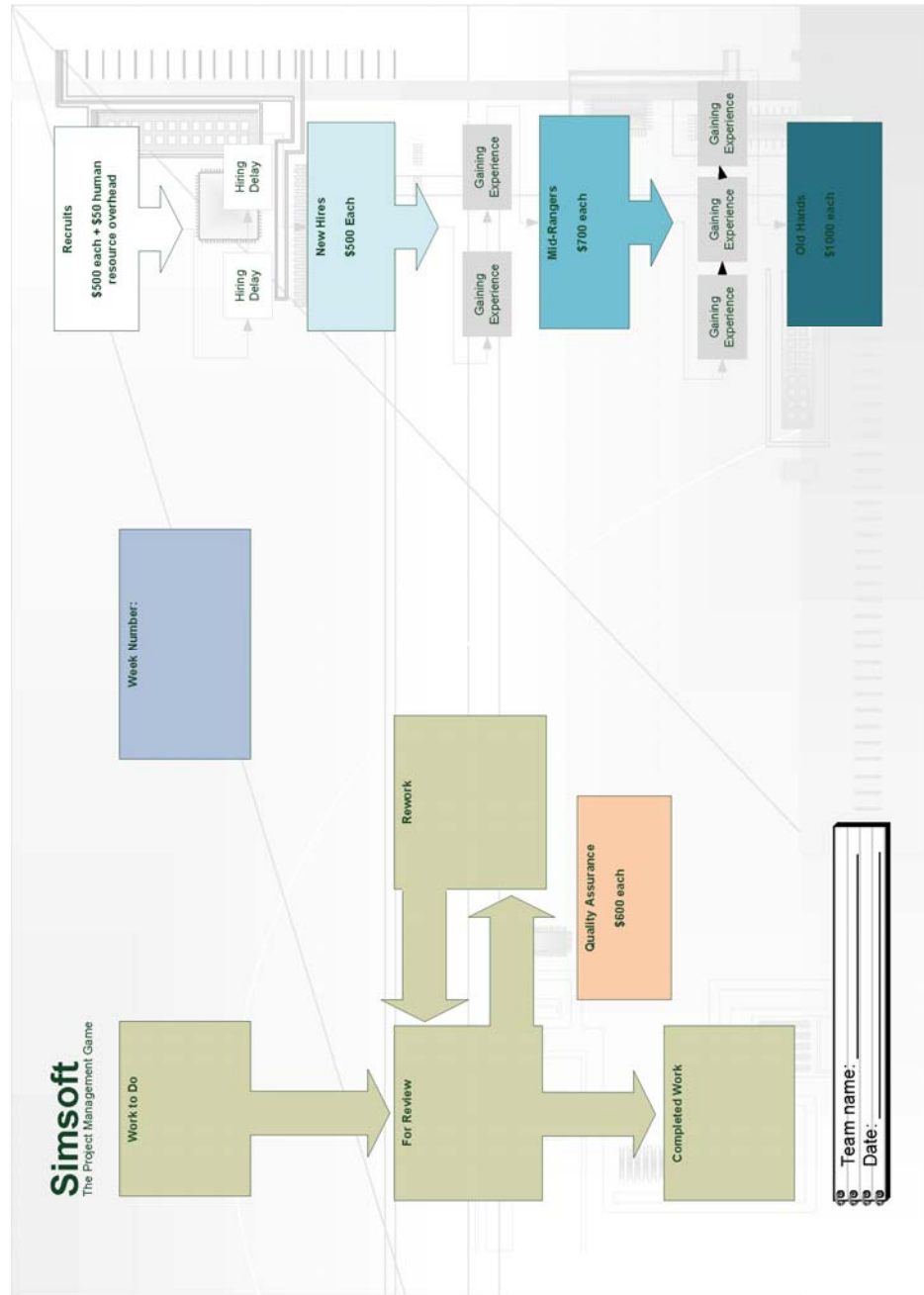
Figure 16: Predator and prey oscillations.

The stock-and-flow notation of this diagram, first described by Forrester (1961, pp. 67 - 72), consists of a small number of symbols that together form a grammar telling a story:

- Stocks or levels can be thought of as nouns since they represent an accumulation of something (hares, lynxes, money, inventory, staff, morale...) at a point in time.
- Flows or rates determine how the stocks will be filled or drained and so are analogous to verbs. Stuff (hares, lynxes, money, material, people, orders...) flows through the pipe of the flow in the direction of the arrow and at a rate determined by the flow regulator in the middle. The flow regulator is fitted with a spigot that can be conceptually tightened or loosened by other variables within the model. The cloud at the end of the flow represents the boundary of the model.
- Converters modify flows within the system, just as adverbs modify verbs. They are often used to break out the detail of the logic that might otherwise be buried within a flow and might be used to represent constant values. Converters typically influence the behaviours of the regulators on the flows.
- Connectors tie the other three building blocks together. They represent inputs and outputs, not inflows and outflows. Connectors do not take on numerical values— they merely transmit values taken on by other building blocks.

Beneath these symbols are stored the functions and values (depending on the implementing software) that drive the simulation and ultimately yield the output shown in Figure 16.

Appendix C: Simsoft Game Board



Appendix D: Simsoft Instructions to Players

About Simsoft

Simsoft is a game based on problem-based learning principles that is designed to demonstrate some fundamental principles of software project management in a controlled and inexpensive way so that software engineers and software project managers don't repeat the same mistakes in the real world.

In Simsoft, you play the role of a project manager for a software development company. For each project, you can draw on a pool of software developers, each with different levels of experience, skills, and charge-out rates. You will be presented with a scenario for a software development project, which you will then manage through to delivery. A business analyst and technical architect from your company have already worked with your client to flesh out the requirements of the project and a high-level technical architecture. The results are in an attached project document. Your job as project manager will be to:

- Read the requirements and come up with time and cost estimates.
- Decide on the number and composition of your software development team.
- Manage the project through to completion.

Don't be too concerned with specific technologies such as programming languages or databases, or with re-engineering business processes: these have already been worked out by the business analyst and technical architect in discussions with the client.

Good luck!

Playing Simsoft

To play Simsoft you will need to read and analyse the scenario and the project documentation and then become familiar with the model behind Simsoft. This is a system dynamics model but the causal loop model over the page gives a high-level view of this.

The basic cycle of play is:

-
- Review the state of the game from the board and decide if any adjustment is needed to the project's personnel. If you feel you need to hire more staff, use your poker chips to buy an appropriate piece from the game administrator. The administrator will place the piece on the board in the correct place.
 - You should also review the project report panel of the Calculator (Section One) to see if anything else needs to be considered. There may be surprise project events popping up that you will need to factor into your decisions.
 - If your team has decided to make changes, enter the changes into the calculator (Section Two). Press the Submit button even if you have decided to make no changes.
 - Section 3 of the calculator will tell you which pieces to move around the board.
 - Press the Advance key to move the project on by one week.
 - The game is now in a new state and the cycle begins again.

Feel free to use the game board as your scratch pad for any notes and thoughts you may have during the game.

You can ask your administrator for any help during the course of the game.

Statement of Work

Your new client is Musty Books.

Musty Books is a second-hand book retailer that two years ago started selling its books online. Now, online sales exceed over-the-counter sales by a ratio of 4:1. Online customers normally pay for their online purchases by credit cards, but the commission fees charged by Musty Books' credit card provider are becoming expensive. Also, some customers don't have credit cards or are wary about buying online. While Musty Books allows these customers to order online and then post a cheque or money order, the process of matching mailed cheques to online orders, sometimes separated by a couple of weeks, is cumbersome and error-prone.

To streamline its order processing and to help its cashflow, Musty Books has decided to offer its customers the option of paying for their purchases by BPAY, a centralised bill paying system in which all the major Australian banks and credit unions participate. Under this scheme, customers will be given Musty Books' BPAY biller reference number and an order number in the final stage of the

ordering process. Later, they can then log onto their own bank's web site and pay for their purchases by quoting these two numbers. After a couple of days, the funds will be credited to Musty Books' bank account and Musty Books' bank, Robba Bank, will send them a daily comma-separated text file detailing the amounts and order numbers.

Robba Bank won't email the potentially sensitive BPAY file to its clients and won't allow clients to connect to its own systems to collect the file. Instead, Robba Bank will install a small application, FileSucker, on one of Must Books' computers which will periodically connect to the bank and securely download the BPAY file. Once the file has been downloaded, Musty Books can process it to fulfil any outstanding orders and reconcile its bank account.

The necessary paper work has been completed with the bank and Musty Books has received its BPAY biller number and a copy of the FileSucker software. Changes have been made to the online ordering process to offer the new payment option and these will be turned on once the backend processing is complete.

Your job is to manage the project and see that it is delivered according to the statement of work, which covers the installation and configuration of the FileSucker software and automating the collection and processing of the BPAY file. Musty Books have told you they plan to relaunch their online store and its new payment option in exactly four months time and they have already committed to expensive print and television advertising. They also believe their online store and the BPAY option will become the company's major source of income and a vital part of its cashflow so the speed and accuracy of processing are critical.

Appendix E: Simsoft Instructions to Game Administrators

For each game session, the following materials are needed:

- Simsoft game boards
- Marker pens
- Note pads and pencils
- Laptops or PCs for the calculator, including spare extension cords
- Poker chips
- Unit-of-work counters
- Personnel pieces
- Statements of work
- Instructions to participants

Set up the laptops or PCs as close as possible to the related game board. Start the calculator by double clicking on the SimsoftCalc.bat file in the *bin* directory.

Ask the players to complete the pre-game survey as soon as they come in.

When all players have completed the pre-game survey and are ready to start playing, ask them to form into teams of two or three or more. If possible rearrange the teams so they aren't composed of all the same skills sets (all developers, all students, or all project managers.)

Give a brief introduction to the game session and how things will play out.

Hand out the player instructions and briefly describe the game scenario. Ask the players to read it together and discuss before they start playing the game. Describe how the game will work by referring to the player instructions. Reiterate that the basic cycle of play is:

- Review the state of the game from the board and decide if any adjustment is needed to the project's personnel. If they want to hire more staff, they need to use their poker chips to buy a piece from the game administrator. They should also review the project report panel of the Calculator (Section One) to see if anything else needs to be considered.

-
- If the team has decided to make changes, the changes should be entered into the calculator (Section Two). Press the Submit button even if no changes are made.
 - Move the unit of work pieces and personnel pieces around on the board according to Section 3 of the calculator.
 - Press the Advance key to move the project on by one week.
 - The game is now in a new state and the cycle begins again.

Encourage the players to use the game board as their scratch pad for notes and thoughts as they play the game.

While the game sessions are in progress, keep an eye out for any players hanging back and not participating or who are being overridden or dominated by other players. Encourage them to participate in the game by, for example, asking them what they thought of the last decision.



Information Letter for Participants

Research Project: Games and Software Engineering

The aim of this research project is to determine if serious games (games for the purpose of teaching and learning) can contribute to better software engineering project management.

This research project is being undertaken as part of the requirements of a PhD at Edith Cowan University and has been approved by the ECU Human Resources Ethics Committee.

The Game Sessions

Participants will be formed into small teams of two to five people and will be given a simulated software project to operate from start-up to the delivery of its objectives. Based on the starting scenario of the game, information provided during the game, and their own real-world experience, the volunteers make decisions about how to proceed— whether to hire more staff or reduce the number, whether to purchase certain equipment, what hours should be worked, and so on. After each decision set (one set per week) has been entered by the players through an on-screen 'dashboard', the game is run for its next time period. The game is now in a new state which the players must interpret from the reports or dials or other interfaces the game provides. A fresh set of decisions is entered and the life of the simulated project continues.

The game will be overseen by an administrator (the principal researcher) whose role will be to:

- Explain the learning objectives to the participants. The participants will be made familiar with the decision-making environment created by the game, the type of decisions that will be required, and the quantifiable indicators of effective decision making.
- Provide the participants with feedback and technical assistance during the decision rounds.
- Run an after-game debrief that helps relate elements of game play to broader software engineering principles.

Because group interaction is an important part of the decision making process, an audio recording of the game sessions will be made.

Participation is entirely voluntary and withdrawing at any time or deciding not to participate will have no consequences.

Duration

The workshop will run for about an hour, including time to complete a short, post-game questionnaire.

Confidentiality

Data collected during the game sessions will help determine what value serious games have in software engineering education and the results will be published in conference papers and journal articles. None of the data collected during the sessions will identify individual participants.

Only the principal research and project supervisors will have access to the data. Paper records will be kept in a locked filing cabinet in a locked office on the ECU Mt Lawley



campus. Electronic records will be stored on a password-protected removable hard drive, stored in the same office. These records will be kept for the time required to complete the research and the requirements of the PhD. Afterwards, the paper records will be shredded and disposed of on a secure collection bin, while the electronic records will be erased using a secured deletion application.

What To Do Next

If you would like to participate in the game sessions, you just need to read and sign the Informed Consent document and bring it along it along to one of the game sessions.

Any Concerns or Questions?

If you have any questions or require any further information about the project, please contact:

Craig Caulfield (principal researcher)
Faculty of Computing, Health and Science
ECU Mt Lawley Campus
MT LAWLEY WA 6056
Email: ccaulfie@student.ecu.edu.au

Dr Paul Maj (project supervisor)
Faculty of Computing, Health and Science
13.135 ECU Mt Lawley Campus
MT LAWLEY WA 6056
Phone: (08) 9370-6277
Email: p.maj@ecu.edu.au

If you have any concerns or complaints about the research project and wish to talk to an independent person, you may contact:

Research Ethics Officer
Edith Cowan University
100 Joondalup Drive
JOONADALUP WA 6027
Phone: (08) 6304-2170
Email: research.ethics@ecu.edu.au

Appendix G: Informed Consent Document



Informed Consent Document

Research Project: Games and Software Engineering

The aim of this research project is to determine if serious games (games for the purpose of teaching and learning) can contribute to better software engineering project management.

The Game Sessions

Participants will be asked to complete a series of tasks as part of a computer game about software engineering project management, complete a short questionnaire, and participate in an after-game debrief that helps relate elements of game play to broader software engineering principles.

Because group interaction is an important part of the decision making process, an audio recording of the game sessions will be made.

Duration

The workshop will run for about an hour, including time to complete a short, post-game questionnaire.

Confidentiality

Data collected during the game sessions will help determine what value serious games have in software engineering education and the results will be published in conference papers and journal articles. None of the data collected during the sessions will identify individual participants.

Only the principal research and project supervisors will have access to the data. Paper records will be kept in a locked filing cabinet in a locked office on the ECU Mt Lawley campus. Electronic records will be stored on a password-protected removable hard drive, stored in the same office. These records will be kept for the time required to complete the research and the requirements of the PhD. Afterwards, the paper records will be shredded and disposed of on a secure collection bin, while the electronic records will be erased using a secured deletion application.

Any Concerns or Questions?

If you have any questions or require any further information about the project, please contact:

Craig Caulfield (principal researcher)
Faculty of Computing, Health and Science
ECU Mt Lawley Campus
MT LAWLEY WA 6056
Email: ccaulfie@student.ecu.edu.au

Dr Paul Maj (project supervisor)
Faculty of Computing, Health and Science
13.135 ECU Mt Lawley Campus
MT LAWLEY WA 6056
Phone: (08) 9370-6277
Email: p.maj@ecu.edu.au

Consent to Participate

I agree to participate in this study, realising I can withdraw at any time. Participation is entirely voluntary, and withdrawing at any time or deciding not to participate will have no consequences. I agree that the research data gathered for this study may be published in conference papers, journal articles and reports provided I am not identifiable. No names or student numbers will be used hence, individual volunteers cannot be identified from the results.



I have read the information above
and any questions I have asked have been answered to my satisfaction.

Signature of volunteer.....

Date.....

Appendix H: Simsoft Database Design

Game transactions are captured in a simple event-based relation data model (Figure 17).

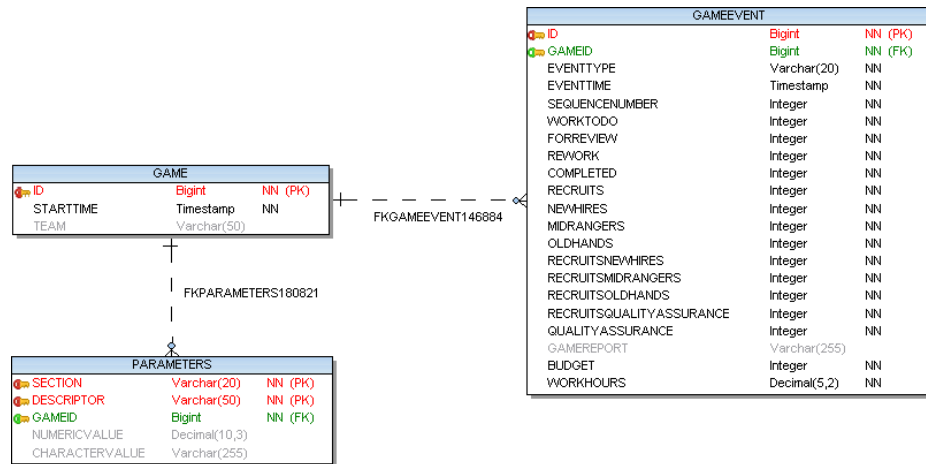


Figure 17: Simsoft data model

Appendix I: Pre-Game Survey

The following are the questions participants answered for the pre-game survey.

Page 1. Introduction

Dear participant,

Shortly you are going to play a game called Simsoft, in which you'll manage a simple software project from inception to completion.

This quick pre-test questionnaire will ask you some general questions about project management and software development so that we have a baseline against which to measure the effectiveness of Simsoft.

Don't worry if you don't know the answer to a question– there's an option on each question to cover that.

Page 2. About You and Your Team

Tell a little about yourself and the team you played with. As mentioned in the Informed Consent and Information to Participants letters, we won't be able to identify individuals from this information.

1. What was the reference number for your team? (This is in the title bar of the calculator and will look something like 21FA-61B2-2F5).

2. Select the option that best describes your current position and the number of years you've been doing it.

Current Position	Experience
Software developer	0 – 1 year
Project Manager	2 to 5 years
General manager	5 to 10 years
Student (under-graduate)	10 to 15 years
Student (post-graduate)	15 or more years.

Page 3. About Project Management and Software Engineering

3. Software engineering is best described as:

-
- The practice of designing, building, and maintaining off-the-shelf software from prefabricated parts.
 - The practice of designing, building and maintaining ad-hoc software without the use of formal methods.
 - The practice of designing, building and maintaining reliable and cost-effective software using standard techniques.
 - The practice of designing, building and maintaining fast and flexible software specifically for Engineering applications.
 - The practice of designing, building and maintaining flashy, cheap and buggy software engineered to generate large initial sales and an on-going market for updates.
 - Don't know.

4. The software crisis is:

- How expensive software is to develop.
- How long it takes to build software.
- How hard software is to write.
- How quickly software becomes obsolete.
- All of the above.
- Don't know.

5. The software crisis exists because:

- Programmers are lazy and managers are ignorant.
- There is as yet no proven scientific method for building robust, efficient, reliable and cost-effective software.
- There can never be a proven scientific method for building robust, efficient, reliable and cost-effective software.
- There are proven scientific methods for building robust, efficient, reliable and cost-effective software, but they are too difficult for most software developers to understand.
- There are proven scientific methods for building robust, efficient, reliable and cost-effective software, but they are being suppressed by the multinational software development conglomerates, who rely on selling annual software updates and bug-fixes.

-
- Don't know

6. Which form of software development model is most suited to a system where all the requirements are known at the start of a project and remain stable throughout the project:

- Waterfall model
- Incremental model
- Evolutionary model
- Spiral model
- Don't know

7. A milestone in project management indicates:

- The passing of 50% of the time allocated to the project.
- The completion of the project
- The conclusion of an important stage of a project and has zero time duration.
- The conclusion of an important stage of a project and has a timer duration equal to the sum of the time durations for each step of that stage.
- Don't know.

8. You find that your project is going to miss its deadline by two months. What action do you take:

- Meet with the client and negotiate to have the team size doubled until the project is back on track.
- Meet with the client and negotiate a reduced scope or a new deadline.
- Do nothing for now and review progress again in a week.
- Keep quiet and ask the team to work over-time to make sure the original deadline is met.
- Don't know.

9. The correct order of steps to solve a problem is:

- Analyse, design, develop, test, evaluate, implement, document.
- Analyse, design, test, develop, document, implement, evaluate.

-
- Design, analyse, develop, document, implement, test, evaluate.
 - Analyse, design, develop, test, document, implement, evaluate.
 - Don't know.

10. The success rate for software projects is very low. Although viewed from different perspectives, studies in the 1990s by Caper Jones, The Standish Group and the Defense Science Board conclude that the success rate for software projects is very low and that the high level of software scrap and rework is mostly indicative of:

- Lack of software development skills
- Immature process
- Lack of adequate schedule
- Lack of adequate budget

Page 4. Thank you

Thank you for taking the time to complete this questionnaire. When all of your teams mates have also finished, let the game facilitator know and they will start you on the game.

Appendix J: Post-Game Survey

The following are the questions participants answered for the post-game survey.

Page 1. Simsoft and Problem-Based Learning Evaluation

Dear participant,

Recently, you played Simsoft, a game about managing a development project.

The way in which this was treated is often called problem-based learning (PBL for short), and is somewhat different to the way of teaching and learning that you are probably familiar with. This means rather than giving people knowledge and then giving them some practice to apply the theory afterwards (such as tutorials, laboratory exercises and so on), it gives participants a problem to examine first and then goes through a solution later, but only after they have had time to digest the problem and discuss it amongst themselves.

We are interested to know your views about this learning approach. Please therefore, take a few minutes to answer the following questions.

Page 2. About You and Your Team

Tell a little about yourself and the team you played with.

As mentioned in the Informed Consent and Information to Participants letters, we won't be able to identify individuals from this information.

1. What was the reference number for your team? (This is in the title bar of the calculator and will look something like 21FA-61B2-2F5).

2. Select the option that best describes your current position and the number of years you've been doing it.

Current Position

Software developer

Project Manager

General manager

Student (under-graduate)

Student (post-graduate)

Experience

0 – 1 year

2 to 5 years

5 to 10 years

10 to 15 years

15 or more years

Page 3. About Games in General

These questions are about games, like Simsoft, in general.

3. Did you find the game interesting?

- Very Boring
- Boring
- Neutral
- Interesting
- Very interesting

4. Did you find that working on a detailed problem scenario helped put some aspects of project management or software engineering theory into context?

- Not at all
- Not really
- Neutral
- Somewhat
- Very much

Any other comments?

5. Do you agree with the following statement:

"Games are a better way to learn and understand technical material than more conventional ways such as through books, case studies, or lectures"

- Not at all

-
- Not really
 - Neutral
 - Somewhat
 - Very much

Any other comments?

6. Do you agree with the following statement:

"Do you think that games are a way of more thoroughly learning a topic than more conventional means (books, case studies, lectures)."

- Not at all
- Not really
- Neutral
- Somewhat
- Very much
- Other (please specify)

Page 4. About Simsoft in Particular

The following questions ask for your thoughts about playing Simsoft and whether it was easy to use and understand.

7. Have you enjoyed playing the game?

- Not at all
- Not really
- Neutral
- Somewhat
- Very much

Any other comments?

8. In general, did you find the task of playing Simsoft:

- Too easy
- Easy
- About right

-
- Hard
 - Very hard
 - Other (please specify)

9. Were the written instructions for Simsoft easy to understand?

- Very difficult
- Difficult
- Neutral
- Easy
- Very easy

Any other comments?

10. Was Simsoft easy to use and navigate around?

- Very difficult
- Difficult
- Neutral
- Easy
- Very easy

Any other comments?

11. Do you think the scenario represented something you might encounter in the real world?

- Not at all
- Not really
- Neutral
- Somewhat
- Very much

Any other comments?

12. Was the logic of the game play apparent from the design of the game board?

- Not at all
- Not really

-
- Neutral
 - Somewhat
 - Very much
 - Other (please specify)

13. Was it useful to play the game as a team?

- Not at all
- Not really
- Neutral
- Useful
- Very useful

Any other comments?

14. Simsoft comes in two parts: the game board and a small calculator in which you entered your decisions. Would you have preferred a fully computerised version?

- Definitely not
- Not really
- Neutral
- Yes
- Positively yes
- Other (please specify)

15. Did you think the length of time you played the game was:

- Far too short
- Too short
- About right
- Too Long
- Far too long
- Other (please specify)

16. Considering the amount of time you spent playing Simsoft, would you be prepared to spend more or less time playing future, more comprehensive versions?

-
- Much less time
 - A little less time
 - About the same amount of time
 - A little more time
 - Much more time
 - Other (please specify)

17. Simsoft is an example of a 'serious game' or a game used for learning rather than pure entertainment. Have you played any other serious games?

- Yes
- No

18. If you have played other serious games, how do you think Simsoft compares?

- Much worse
- Slightly worse
- About the same
- Better
- Much better
- Any other comments?

19. How do you think Simsoft could be improved?

Page 5. Test Your Knowledge

Just before you started playing Simsoft, you completed a short survey that asked some general project management and software engineering questions. The following are some similar questions. Use your general knowledge and anything you learned during the game to answer these.

Don't worry if you don't know the answer to a question– there's an option for that.

20. What is the single largest computer-related cost for most organisations?

- Software analysis and design.

-
- Software implementation.
 - Software testing.
 - Software maintenance.
 - Coca Cola and pizza.
 - Don't know.

21. Which of the following is the logical flow of any project?

- Planning, Initiating, Executing, Controlling, Closing
- Initiating, Planning, Controlling, Executing, Closing
- Planning, Initiating, Executing, Controlling, Closing
- Initiating, Planning, Executing, Controlling, Closing
- Don't know.

22. "Analysis requires the software engineer to become 'consciously expert' in the domain". This means:

- The software engineer has to be conscientious about how they deal with experts.
- The software engineer has to have a good and trained mind (i.e. an "expert consciousness")
- The software engineer has to learn what to do in the domain, without thinking about how that knowledge was achieved.
- The software engineer has to learn what to do in the domain, and be aware of what it is that they are doing.
- The software engineer doesn't have to learn what to do in the domain, because it is enough to identify those experts who already do.
- Don't know.

23. Most common cause of conflicts in a project is:

- Schedules
- Technical opinions
- Personal Issues
- Project priorities
- Don't know

24. Your project has just been assigned some critical extra work that must be completed by the original deadline in two month's time. What action do you take?

- Calculate the number of extra people required and negotiate with the client to increase the size of the team accordingly.
- Negotiate with the client to see what features can be deferred until later in order the fit in the new critical work.
- Refuse to accept the work because it is outside the agreed scope.
- Don't know.

25. Product quality can be defined as:

- Delivering a product with correct requirements
- Delivering a product using correct development procedures
- Delivering a product which is developed iteratively
- Delivering a product using high quality procedures
- Don't know.

26. Which form of software development model is most suited to a system where the requirements are still being defined and the client is very concerned about the overall development cost.

- Waterfall model
- Incremental model
- Evolutionary model
- Spiral model
- Don't know

27. The project manager of a large multi-location software project team has 24 members, out of which 5 are assigned to testing. Due to recent recommendations by an organisational quality audit team, the project manager is convinced to add a quality professional to lead the test team at additional cost, to the project.

The project manager is aware of the importance of communication, for the success of the project and takes this step of introducing additional communication channels, making it more complex, in order to assure quality levels of the project.

How many additional communication channels are introduced as a result of this organizational change in the project?

- 25
- 24
- 5
- 1

Page 6. Anything Else?

28. Have we missed anything you think we should know about? Please let us know.

Page 7. Thank you

Thank you for taking the time to complete this questionnaire. All of your answers are completely confidential.

Appendix K: Simsoft Finding Review

The following are the questions four random participants were asked answer in order to confirm the findings described in Chapter 4— Findings.

Page 1. Simsoft Findings Review

Dear participant,

Recently, you played Simsoft, a game about managing a development project. You would be aware that Simsoft is being used as part of a research project to see if and how games can be used as part of software engineering and project management education. A preliminary set of findings have now been determined and as one of the original participants of the game sessions, we would like your views on whether the findings seem reasonable and are in line with your experience of playing the game.

Feel free to add any further comments.

1. The main finding of the project was that there was evidence the participants were learning by doing and building their own mental models about what was happening. Also, all groups of participants (students, software developers, and project managers) increased their scores between the pre- and post-game surveys.

- Do you think this finding was reasonable?
- No
- Yes
- Other (please specify)

2. The second finding was that games such as Simsoft are not sufficient learning vehicles by themselves and need to be supplemented by other methods. The software developers and project managers were able to make decisions based on experience or completed university studies, but many students said they needed to know more than the game provided.

- Do you think this finding is reasonable?
- No
- Yes

-
- Other (please specify)

3. The third finding was that Simsoft is a suitable pedagogical device for participants of different skills and backgrounds. The participants in this research project came from a variety of Western and Eastern cultures; there were differences in language abilities; and experience in their fields ranged from nothing to seasoned professionals with a wide breadth of work and life experience. Yet, a majority of participants said they found the game interesting and it was pitched at the right level and was something they could easily play and understand.

- Do you think this finding is reasonable?
- No
- Yes
- Other (please specify)

4. The fourth finding was that a majority of participants said they would be prepared to invest greater time and effort in games such as Simsoft if the reward was deeper understanding of a problem domain. Many participants said the game ended too soon or that they would like to create a scenario similar to their own work place or that they wanted more time to discuss their decisions. A group of ten players had previously played a real-time stock market game and felt that games run in real time gave time for considered judgments and added verisimilitude.

- Do you think this finding is reasonable?
- No
- Yes
- Other (please specify)

5. The fifth finding was that the majority of the participants found working in groups was a positive experience. It has already been mentioned that the participants were a diverse group of cultures, skills, and experience and many felt they were able to work out collaborative decisions in a constructive manner. However, as with any group activity, facilitators need to be cognisant of any individuals dominating a group or others who might need a gentle prompt to contribute more.

-
- Do you think this finding is reasonable?
 - No
 - Yes
 - Other (please specify)

6. The last finding was a majority of participants preferred to play around a game board rather than a fully computerised game because this was a familiar and simple activity and less time was lost to overcoming technological problems and to making simple ergonomic arrangements such as fitting all the team around a single computer. Even so, facilitators need to prepare the participants for the game sessions by giving them clear instructions and sufficient lead time to absorb the information.

- Do you think this finding is reasonable?
- No
- Yes
- Other (please specify)

Appendix L: Full Data Extract of Games Used in Software Engineering Education

ID	Study	Description	Experimental Design	Sample Size (if known) and Population	Data Collection Tool	SWEBOK Knowledge Area	Bloom Learning Outcome	Observed Learning Outcomes
GS-01	University/Industry Collaboration in Developing a Simulation Based Software Project Management Training Course. (Collofello, 2000).	A single-player game, based on a system dynamics model with an iThink user interface that models a software project. Players attempt different management exercises (risk management, life cycle model comparison, critical path scheduling, etc.) that follow the lecture material.	Non-experimental	16 students	Questionnaire	Software engineering management Software engineering process	Knowledge	Learning was not assessed.
GS-02	¹ Quantitative Modeling for the Interactive Simulation of Software Project (Drappa & Ludewig, 1999) ² Simulation in Software Engineering (Drappa & Ludewig, 2000)	SESAM (Software Engineering Simulation by Animated Models) is a model of a software project. Users run the model loaded with its initial project state and then tweak it to simulate different scenarios before running it again. Players take the role of a project manager and must plan and control a simulated project. Rather than a graphical user interface, players control the game by typing commands in a modelling language. Players analyse their performance through an after-game analysis tool.	¹ Non-experimental ² True Experimental	¹ 10 undergraduate project management students ² 19 second-year computer science students	¹ n/a ² Pre- and post-game tests Project plan	Software engineering management	Knowledge	¹ A qualitative assessment that the players experienced something similar to a real project, including panic when the deadlines were approaching. ² Students in the experimental and control groups improved their performance in successive game sessions.
GS-03	An Interactive Multimedia Software	Case studies are presented through a simulated office environment and then	Non-experimental	Post-graduate distance	Questionnaire	Software requirements	Knowledge	Learning was not assessed.

	House Simulation for Postgraduate Software Engineers (Sharp & Hall, 2000)	completed outside of the game environment.		education software engineering students.		Software design Software construction Software testing		
GS-04	How to Successfully Use Software Project Simulation for Educating Software Project Managers (Mandl-Striegnitz, 2001)	Participants play two sessions of SESAM (GS-02) and their tutor analyzed their performance and provided feedback in between.	Non-experimental	40 undergraduate software engineering students	Questionnaire	Software engineering management	Knowledge	Players improved their performance in the second session but still had problems monitoring their project and tracking progress.
GS-05	An Experiment for Evaluating the Effectiveness of Using a System Dynamics Simulation Model in Software Project Management Education (Pfahl et al., 2001)	A three-phase (design, implementation, test) waterfall project modeled using System Dynamics. Key project variables were project duration, effort consumption, product size, and quality after testing. Participants were separated in two groups: one group managed a simulated software project with the aid of a System Dynamics model (Abdel-Hamid, 1989); the other group used COCOMO (Boehm et al., 2000).	True Experimental	12 post-graduate software engineering students	Pre- and post-test questionnaires	Software engineering management	Knowledge	Pre- and post-session surveys indicated that participants were improving their knowledge of project management patterns and behaviors. Those using the simulation models performed better than those using COCOMO.
GS-06	An Externally Replicated Experiment for Evaluating the Learning Effectiveness of Using Simulations in Software Project Management Education (Pfahl et al.,	Same as for GS-05.	True Experimental	¹ 12 graduate and post-graduate students majoring in computer science.	Pre- and post-test questionnaires	Software engineering management	Knowledge	The results confirmed the initial findings in which students using the System Dynamics model generally performed better in the pre- and post-test questionnaires than those using COCOMO.

	2003). Evaluating the Learning Effectiveness of Using Simulations in Software Project Management Education: Results From a Twice Replicated Experiment (Pfahl et al., 2004).			² 13 senior under-graduate students majoring in computer science, electrical engineering, and computer engineering.				
GS-07	Problems and Programmers: An Educational Software Engineering Card Game (Baker et al., 2003). An Experimental Card Game for Teaching Software Engineering Processes (Baker et al., 2005). Teaching Software Engineering Using Simulation Games (Navarro et al., 2004).	A competitive card game called Problems and Programmers in which students play the role of project manager in a waterfall project. All players lead the same project. Players must balance several competing concerns including budget and the client's demands regarding the reliability of the final software. Who finishes first, wins.	Non-experimental	28 undergraduate students who had completed an introductory software engineering unit	Questionnaire	Software engineering managementSoftware engineering process	Knowledge	Players self-assessed their level of learning in a post-game survey. Most said the game was not good at teaching new knowledge or reinforcing existing knowledge.
GS-08	Engendering an Empathy for Software	Players act as a project manager to deliver a product within time and budget	Non-experimental	Undergraduate software	Post-test questionnaire	Software engineering	Knowledge	The degree of learning was self-assessed by the participants and

	Engineering (Shaw & Dermoudy, 2005).	constraints. SimjavaSP uses discrete-event simulation as the game engine.		engineering students		management		was found to be positive.
GS-09	Model-Driven Game Development: Experience and Model Enhancements in Software Project Management Education (Barros et al., 2006) A Simulation-Based Game for Project Management Experiential Learning (Dantas et al., 2004).	Uses simulation to support decision-making on software project management. In the game, The Incredible Manager, the player sets project parameters such as staffing and work hours and executes the project for a period of time. The simulation can be stopped so the parameters can be tweaked.	Non-experimental	7 post-graduate students in a software project management course, 8 undergraduate and post-graduate students from a software development laboratory, and 9 other undergraduates.	Questionnaire	Software engineering management	Knowledge	Players self-assessed their level of learning in a post-game survey. Most said they had learned something new but only one person completed their project successfully.
GS-10	SimVBSE: Developing a Game for Value-Based Software Engineering (Jain & Boehm, 2006).	Focused on value-based software project management: every requirement, use case, object, test case and defect is treated as equally important; earned value is used to track project cost and schedule; a separation of concerns is practiced, in which the responsibility of software engineers is confined to turning software requirements into verified code. The player's avatar visits different game rooms and collects information from various stakeholders about the current project and how to proceed. Still not fully implemented	n/a	n/a	n/a	Software engineering management	Knowledge	n/a

GS-11	SimSE: A Software Engineering Simulation Environment for Software Process Education (Navarro, 2006).	Same as for GS-14.	True Experimental	19 undergraduate software engineering students	Pre- and post-test questionnaires	Software engineering management Software engineering process	Knowledge	All groups improved their knowledge, but those in the control groups outperformed those who had played SimSE in the post-test. When players play SimSE for longer periods, their scores improved. But, many dropped out due to boredom or frustration before this point.
GS-12	e-Learning in Project Management Using Simulation Models: A Case Study Based on the Replication of an Experiment (Rodriguez et al., 2006).	A replication of the GS-05	True Experimental	11 second-year undergraduate students taking a software engineering module	Pre- and post-test questionnaires	Software engineering management	Knowledge	According to the post-test and qualitative results, students using the simulation appear to have understood the software engineering principles it was trying to teach better than those in the control group
GS-13	Using Games in Software Engineering Education to Teach Risk Management (Taran, 2007).	A competitive board/card game that focuses on risk management. Players take the role of project manager and have to develop a product and sell it in the market. The player with most money at the end wins. A dice is used to simulate eventuated risk events.	Non-experimental	150 on-campus and distance students.	5-question questionnaire	Software engineering management	Knowledge	Players said they understood the learning objectives of the game. The degree of learning was not assessed.
GS-14	Towards Game-Based Simulation as a Method of Teaching Software Engineering (Oh & van der Hoek, 2002).	A single-player game for multiple development methodologies (waterfall, RUP, rapid prototyping) in which the player takes the role of a project manager leading a team of developers. The team must complete a virtual software project by hiring staff, assigning	Non-experimental	29 undergraduate software engineering students	Post-test questionnaires	Software engineering management Software engineering process	Knowledge	Players felt the game reinforced what they already knew but provided little new knowledge. Players are demonstrating aspects of learning theories such as learning by doing, situated learning,

	<p>Design and Evaluation of an Educational Software Process Simulation Environment and Associated Model (Navarro & van der Hoek, 2005).</p> <p>SimSE: A Software Engineering Simulation Environment for Software Process Education (Navarro, 2006).</p> <p>Comprehensive Evaluation of an Educational Software Engineering Simulation Environment (Navarro & van der Hoek, 2007).</p>	<p>tasks, monitoring progress, purchasing resources.</p> <p>At the end of the game the player receives a score and can analyse their results with an explanatory tool.</p>						<p>discovery learning, learning through failure, and Keller's ARCS.</p> <p>SimSE is most effective when used with other teaching methods.</p>
GS-15	<p>Enhancing Software Engineering Education Using Teaching Aids in 3-D Online Virtual Worlds (Ye et al., 2007).</p>	<p>Two exercises were performed in Second Life, an online virtual environment.¹ Groupthink exercise: groups of students are given a software specification and must reach a design consensus. Afterwards, individuals are asked</p>	Non-experimental	<p>¹ 29 undergraduate students</p> <p>² 26 undergraduate</p>	Questionnaire	Software engineering processSoftware requirementsSoftware engineering management	Comprehension	<p>Most students said the exercises helped them understand the fundamentals of software specification activities and the principles of software development processes.</p>

		questions about the specification and points are awarded for correct answers. ² SimSE exercise: the game from GS-14 was modified to run in Second Life.		students				
GS-16	Requirements Game: Teaching Software Project Management (Zapata & Awad-Aubad, 2007).	Teams of 4 or 5 players take on roles such as project manager, developers, designers, or analysts. For a given case-study, the players must produce documentation such as an ER diagram, sketches of at least 3 GUIs, and an estimation of the effort required, and then build the application in, say, Microsoft Access. A facilitator plays the role of a client giving more instructions or clarifications. Fines may be imposed for time or budget over-runs.	Non-experimental	47 systems engineering undergraduate students. 8 systems engineering Masters students. 30 systems, industrial, and administrative engineering undergraduate students.	Performance in the game alone	Software requirements	Knowledge	Not assessed
GS-17	A Game for Taking Requirements Engineering More Seriously (Knauss et al., 2008).	A web-based game that can be completed in about 10 minutes. Software requirements are visualized as a bag of balls that flow from the customer to an analyst, a designer, and a developer depending on the development process chosen. Alternate flows may be taken (such as the client speaking directly to the developers to clear up misunderstandings), which can change the rate of flow.	n/a	n/a	n/a	Software requirements	Knowledge	Not assessed
GS-18	On the Role of	Same as for GS-14.	Quasi-	11 under-	Observation and	Software	Knowledge	Players demonstrated aspects of

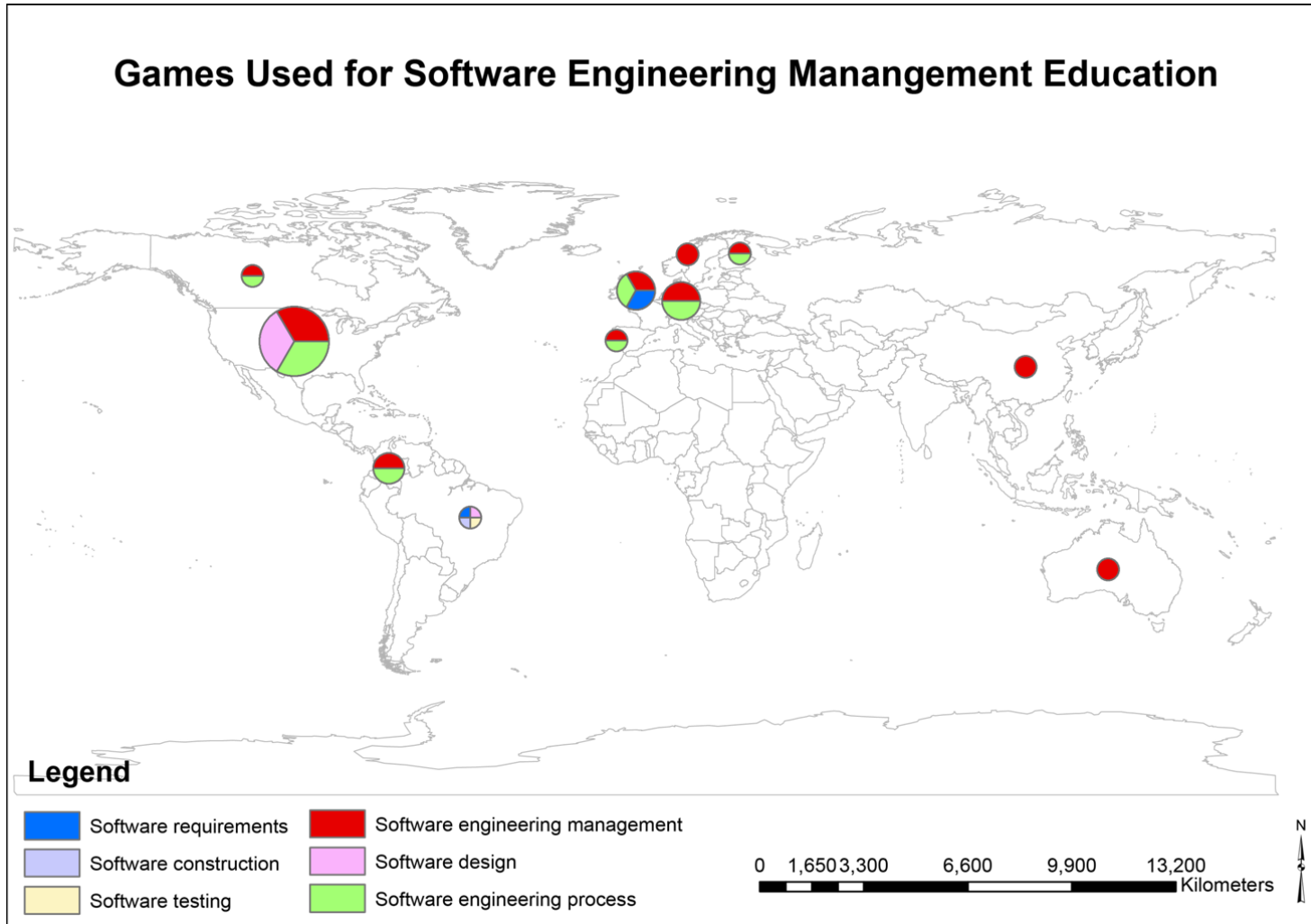
	Learning Theories in Furthering Software Engineering Education (Navarro & van der Hoek, 2008).		experimental	graduate students who had passed an introductory software engineering course.	post-test interview	engineering management Software engineering process		learning theories such as learning by doing, situated learning, elaboration, discovery learning, learning through failure, Keller's ARCS, and learning by reflection.
GS-19	An Evaluation of a Mobile Game Concept for Lectures (Wang et al., 2008).	The lecturer acts as a game show host and students answer multiple choice questions about a particular software design issue through their laptop or mobile phone. Players have to answer correctly to get to the next round. The winner is the last person standing.	Non-experimental	20 software engineering Masters students.	Questionnaire Performance in the game	Software design	Knowledge	Players felt the system made them pay closer attention during the lecture and that they learned more than through a traditional lecture.
GS-20	Multi-Site Evaluation of SimSE (Navarro & van der Hoek, 2009).	Same as for GS-14. SimSE was run in game sessions in which the original game designers were not directly involved.	True Experimental	Site 1: 14 students in a senior research seminar course, most of whom had passed a software engineering course. Site 2: 19 undergraduate software engineering students. Site 3: 48 undergraduate	Post-test questionnaires, performance in SimSE, and final course grades.	Software engineering management Software engineering process	Knowledge	Students seemed to learn the concepts the game is designed to teach. The game was suitable for students of varying abilities and backgrounds. SimSE is most effective when used with other teaching methods.

				software engineering students.				
GS-21	Empirical Evaluation of an Educational Game on Software Measurement (Gresse von Wangenheim et al., 2009).	In X-MED, the player takes the role of a measurement analyst and defines and executes a measurement exercise based on a given development scenario. A score is calculated based on the number of correct decisions made, and the player is presented with an analysis of their performance.	True Experimental	15 computer science post-graduate students	Pre- and posttest questionnaires	Software engineering management Software engineering process	Knowledge	The results don't conclusively point to a positive learning effect, although most players' subjective evaluation was that the game helped them understand the topic.
GS-22	Adapting Game Technology to Support Software Engineering Process Teaching: From SimSE to MO-SEProcess (Zhu et al., 2007). A Software Engineering Education Game in a 3-D Online Virtual Environment (Wang & Zhu, 2009).	A game based on SimSE (GS-14) using the rapid prototyping profile and deployed to Second Life.	Non-experimental	52 software engineering students	A six-question post-test questionnaire.	Software engineering process	Knowledge	Players self-assessed their level of learning in a post-game survey. Most said the game had helped them understand the software development process better.
GS-23	PlayScrum- A Card Game to Learn the Scrum Agile Method (Fernandes & Sousa, 2010).	Focused on the Scrum (Schwaber, 2004) agile development process. Further development of Problems and Programmers (Baker et al., 2005). Played by 2 to 5 people. Cards are used to represent tasks, problems, developers,	Non-experimental	13 post-graduate students.	Questionnaire	Software engineering management Software engineering process	Knowledge	Students improved their performance in successive game sessions. Players analyze their performance through an after-game analysis tool

		and artifacts. The winner is the person who performs all tasks with the least number of errors. A roll of a dice determines the flow of the game.						
GS-24	<p>Evaluation of a Game to Teach Requirements Collection and Analysis in Software Engineering at Tertiary Level (Hainey et al., 2010).</p> <p>An Application of Games-Based Learning Within Software Engineering (Connolly et al., 2007).</p>	<p>Players take on specific roles (project manager, systems analyst, systems designer, team leader). The systems analyst moves their avatar through the game world to collect requirements by asking questions of game characters. When the analyst thinks they have all requirements, they prepare a requirements document and send it to the project manager, who must decide whether to proceed with the project.</p>	True Experimental	<p>55 university students and 37 higher-education students (92 in total). The majority had little or no instruction in requirements collection or analysis.</p>	Pre- and post-test questionnaires	Software requirements	Knowledge	<p>Comparison of pre- and post-game test scores showed an increase in knowledge. Control groups who did not play the game also showed an increase in knowledge. The game was found to be a good supplement to existing courses. Higher education students gained more from the game (better post-game scores) and were more accepting of the teaching technique than further education students.</p>
GS-25	<p>Learning Software Engineering Basic Concepts Using a Five-Phase Game (Rusu et al., 2010).</p>	<p>Players take the role of a requirements engineer in a waterfall development (requirements, design, implementation, testing, maintenance phases) software project. The player's avatar must ask questions of on-screen characters to determine the right requirements. Subsequent phases use arcade-style graphics to kill 'computer bugs' or to 'shoot' answers in a multiple choice quiz.</p>	Non-experimental	<p>Developed by teams of undergraduate software engineering students and used by a class of nine middle and high school students with limited or no computer</p>	Pre- and post-test questionnaires	Software engineering management	Knowledge	<p>Comparing pre- and post-game surveys most participants said they gained a better understanding of software development.</p>

				science background.				
GS-26	A Classroom Game for Teaching Management of Software Companies (Zapata, 2010).	Players take turns in rolling a dice and answering a technical question about software development. If the answer is right, the player's team has the chance to solve a project estimation problem. The team with the most correct responses to the questions and estimation problems wins.	Non-experimental	40 systems engineering students	Post-test questionnaire	Software requirements	Knowledge	Players self-assessed their level of learning in a post-game survey. Most said they had learned something new.

Appendix M: Spatial Distribution of Games Used in Software Engineering Education



Appendix N: Review Studies of the Instructional Effectiveness of Games

ID	Year	Study	Number of Studies	Studies Spanned	Findings
GE-1	1966	Cherryholmes, C. H. (1966). 'Some Current Research on Effectiveness of Educational Simulations: Implications for Alternative Strategies.' <i>American Behavioral Scientist</i> , vol. 10, no. 2 (October), pp. 4 – 7.	6	1963 - 1966	<p>While students appeared to be more motivated and interested in playing games, "there are no consistent or significant differences in learning, retention, critical thinking or attitude change" (Cherryholmes, 1966, p. 6).</p> <p>However, there were side benefits for the developer of the game because they had to understand the decisions, processes, and responses represented by the game in order to build it.</p>
GE-2	1973	Greenblat, C. S. (1973). 'Teaching with Simulation Games: A Review of Claims and Evidence.' <i>Teaching Sociology</i> , vol. 1, no. 1, pp. 62-83.	Many	Up to 1972	<p>Noticed methodological problems in the extant studies, of which there weren't then many. Concluded that it is difficult to compare games to other forms of instruction when the data for these latter is also lacking.</p> <p>Of note: "although there is little evidence that students learn more when taught by games than by conventional methods, there is no evidence that they learn less. In fact, studies of cognitive learning point to "no difference" or differences in favor of games that are not statistically significant." Hence, games seem to be at least as effective as other modes of teaching, and further studies may show yet more significant results." (Greenblat, 1973, p. 80).</p>
GE-3	1973	Greenlaw, P. S. and Wyman, F. P. (1973). 'The Teaching Effectiveness of Games in Collegiate Business Courses.' <i>Simulation & Games</i> , vol. 4, no. 3 (September), pp. 259 – 294	Many	Up to 1972	<p>Although games were popular and widespread, very little true experimental data had been collected. Many games didn't show a marked learning bias in favour of games; some had descriptive data that this was happening but few had hard, statistical data.</p> <p>However, in some games "varied and effective team interaction may be a more effective source of learning than the model itself" (Greenlaw & Wyman, 1973, p. 263).</p>
GE-4	1976	Neuhauser, J. J. (1976). 'Business Games Have Failed.' <i>Academy of Management Review</i> , vol. 1, no. 4 (October), pp. 124 – 129.	6	1966 - 1974	<p>A subjective rather than analytical assessment, Neuhauser believes the early enthusiasm for games was linked in part to access to affordable computers by universities and businesses with which to run the games. However, the games were not engaging enough to sustain players' attention.</p>
GE-5	1977	Pierfy, D. A. (1977). 'Comparative Simulation Game Research: Stumbling Blocks and Steppingstones.' <i>Simulation & Games</i> , vol. 8, no. 2 (June), pp. 255 - 268	22	1963 - 1975	<p>The results for much the same as Cherryholmes'. Pierfy concluded that:</p> <p>"In terms of fostering student learning, simulation games are no more effective than conventional classroom instruction. However, the research also suggested that games appear to have an advantage when it comes to retention of information. In terms of the ability to change attitudes and student interest, the simulation games appear to have an advantage over conventional instruction." (Pierfy, 1977, p. 266).</p>

GE-6	1981	Bredemeier, M. E. and Greenblat, C. S. (1981). 'The Educational Effectiveness of Simulation Games.' <i>Simulation & Gaming</i> , vol. 12, no. 3 (September 1, 1981), pp. 307-332.	Many	1963 - 1980	<p>Even though there was some evidence that material was being remembered by players, hard data that the games were actually effective was difficult to justify. However, affective learning was noticed more than cognitive learning.</p> <p>A more definite evaluation would depend on the following being place: "(1) a theoretically based taxonomy of games with (2) clear theories about (a) what aspects of them are expected to have (b) what sorts of distinct effects (c) on what sorts of students (d) for what reasons. Until these tasks are addressed, we shall probably continue to see results of investigations about "effectiveness"; that are inconsistent, ambiguous, and nondefinitive in support or revision of widespread "impressions" (Bredemeier & Greenblat, 1981, p. 327).</p>
GE-7	1986	VanSickle, R. L. (1986). 'A Quantitative Review of Research on Instructional Gaming: A Twenty-Year Perspective.' <i>Theory and Research in Social Education</i> , vol. 14, no. 3, pp. 245 - 264.	26	Up to 1984	<p>Found that there was little evidence to say that games were more effective than other teaching techniques. However, the way the data was analysed was obscure and involved some mathematical transformations of dubious benefit.</p>
GE-8	1989	Dorn, D. S. (1989). 'Simulation Games: One More Tool on the Pedagogical Shelf.' <i>Teaching Sociology</i> , vol. 17, no. 1, pp. 1-18.	157 books and articles	Up to 1988	<p>In general, players were motivated to play games and interested in the game play, but this did not translate to greater interest in the subject matter. Affective learning was noticed but this was not consistently demonstrated across all studies.</p>
GE-9	1992	Randel, J. M., Morris, B. A., Wetzel, C. D. and Whitehill, B. V. (1992). 'The Effectiveness of Games for Educational Purposes: A Review of Recent Research.' <i>Simulation & Gaming</i> , vol. 23, no. 3 (September), pp. 261-276.	68	1963 to 1984	<p>Of the 68 studies examined, 56% found no difference between games and other forms of instruction; 32% found differences favouring games; 7% favoured games, but the controls were dubious; and 5% found differences favouring conventional instruction.</p> <p>The area in which the game was applied seemed to have some bearing on its effectiveness. For example, 7 out of 8 studies involving maths found that games were superior to traditional instruction. In most studies, students reported more interest in the game than in more conventional instruction.</p> <p>The authors found many of the studies needed to be more rigorous in their experimental design.</p>
GE-10	1994	Dempsey, J., Lucassen, B., Gilley, W. and Rasmussen, K. (1994). 'Since Malone's Theory of Intrinsically Motivating Instruction: What's the Score in the Gaming Literature.' <i>Journal of Educational Technology Systems</i> , vol. 22, no. 2, pp. 173 - 183.	51	Up to 1993	<p>Most of the games were used to teach new skills or practice existing skills, but in many of the studies the learning objectives couldn't be determined.</p>
GE-11	1997	Wolfe, J. (1997). 'The Effectiveness of Business Games in Strategic Management Course Work.' <i>Simulation & Gaming</i> , vol. 28, no. 4 (December), pp. 360 - 376.	Not explicitly stated	1964 - 1990	<p>By far, games were being compared to case studies rather than other pedagogical means. On this basis, "games produced higher learning levels except in studies in which case method protocols were used to determine knowledge scores. In those studies in which the evaluation criteria favored cases, games were equal to the case method in their teaching ability" (Wolfe, 1997, p. 371)</p>

GE-12	1998	Van Joolingen, W. R. and De Jong, T. (1998). 'Scientific Discovery Learning with Computer Simulations of Conceptual Domains.' Review of Educational Research, vol. 68, no. 2, pp. 179 – 201.	Not explicitly stated	Up to 1997	"The general conclusion that emerges from these studies is that there is no clear and univocal outcome in favor of simulations. An explanation of why simulation-based learning does not improve learning results can be found in the intrinsic problems that learners may have with discovery learning." (Van Joolingen & De Jong, 1998, p. 181)
GE-13	1999	Lee, J. (1999). 'Effectiveness of Computer-Based Instructional Simulation: A Meta Analysis.' International Journal of Instructional Media, vol. 26, no. 1, pp. 71- 85.	19	Up to 1999	Many of the studies had confounding variables and experimental design of different quality. Nevertheless, three initial finding were made: 1. If used in presentation mode, hybrid simulations were more effective than the pure simulations; 2. Simulations are almost equally effective for both the presentation and the practice mode if a hybrid simulation was used; 3. students need specific guidance in how to use the game before they start.
GE-14	2004	Gosen, J. and Washbush, J. (2004). 'A Review of Scholarship on Assessing Experiential Learning Effectiveness.' Simulation & Gaming, vol. 35, no. 2, pp. 270-293.	19	1989 - 2004	"Based on Bloom's taxonomy... and rigorous research design standards, there have not been enough high-quality studies to allow us to conclude players learn by participating in simulations or experiential exercises." (Gosen & Washbush, 2004, p. 286).
GE-15	2006	Vogel, J. J., Vogel, D. S., Cannon-Bowers, J., Bowers, C. A., Muse, K. and Wright, M. (2006). 'Computer Gaming and Interactive Simulations for Learning: A Meta-Analysis.' Journal of Educational Computing Research, vol. 34, no. 3, pp. 229-243.	32	Not specified	Those using simulations or games reported higher cognitive gains and more engagement with their own learning than those whose tried traditional pedagogical means. However, the authors had to disregard many other studies because they contained methodological and reporting failings.
GE-16	2011	Sitzmann, T. (2011). 'A Meta-Analytic Examination of the Instructional Effectiveness of Computer-Based Simulation Games.' Personnel Psychology, vol. 64, no. 2, pp. 489 - 528.	65	1976 - 2009	"Overall, declarative knowledge was 11% higher for trainees taught with simulation games than a comparison group; procedural knowledge was 14% higher; retention was 9% higher; and self-efficacy was 20% higher." (Sitzmann, 2011, p. 520). Players gained most from the experience when they were motivated to become actively involved in the game and when the game was embedded in an instructional program rather than a stand-alone exercise.

Appendix O: Peer-Reviewed Conference and Journal Articles Stemming From This Research Project

The following papers based on this research project have been published in peer-reviewed conference papers and journal articles.

Caulfield, C. W. and Maj, S. P. (2001). 'A Case for System Dynamics.' In Z. J. Pudlowski and D. W.-S. Tai (eds.), *Proceedings of the 3rd Asia-Pacific Forum on Engineering & Technology Education*, (Changhua, Taiwan, 8 - 11 July), pp. 49 - 53. Melbourne: UNESCO International Centre for Engineering Education.

Notes: This paper was awarded the UICEE diamond award (first grade) by popular vote of Forum participants for the most significant contribution to the field of engineering education.

Abstract: Engineering education provides a thorough and systematic training in the design, development, maintenance and management of complex technical systems. While such education provides the necessary technical depth to graduates, many technical systems are best understood from the perspective of human and socio-economic relationships. A case in point may be Fred Brooks' law that states adding more developers to a late software engineering project will only make it even more behind schedule. Brooks' law is based on the understanding that additional, new software engineering staff will need time to come up to speed with the project and in doing so will divert the existing developers from their primary tasks. While Brooks' law is intuitively appealing, students and practicing software engineers really have no way of testing its efficacy in their particular situations. A tool to overcome this difficulty may be system dynamics. System dynamics is a systems thinking methodology for building quantitative and qualitative models of complex situations so that they can ultimately be better understood and managed. Accordingly, it can be argued, that system dynamics should be an essential part of the education of engineers from most, if not all, of the major disciplines.

Caulfield, C. W. and Maj, S. P. (2001). 'A Case for Systems Thinking and System Dynamics.' *Proceedings of the 2001 IEEE International Conference on Systems, Man & Cybernetics*, (Tucson, Arizona, 7 - 10 October), pp. 2793 - 2798. Piscataway, New Jersey: IEEE.

Abstract: The title of this paper is too brief to be quite accurate. Perhaps with the following subtitle it does not promise too much: a review of systems thinking that considers its unique history and influences, paradigms and methodologies, and presenting a case for the system dynamics methodology as the best tool for the most diverse range of problem situations.

Caulfield, C. W. (2002). 'A Case for Games in Software Engineering.' In P. Ledington and J. Ledington (eds.), *Proceedings of the 8th Australian and New Zealand Systems Conference*, (Mooloolaba, Queensland), pp. 82 - 94.

Abstract: Computerised management simulation games have been shown to be effective learning tools in a variety of socio-economic and socio-technical environments. Originating in war-gaming and drawing on influences from a range of different fields, games have become established instructional elements in many business, military and educational institutions because of their ability to expand the notional experience of players in a safe, yet challenging, environment. However, they appear to be under represented in the field of software project management. A case is developed that a need and opportunity exists for games in this area.

Caulfield, C. W. and Maj, S. P. (2002). 'A Case for System Dynamics.' *Global Journal of Engineering Education*, vol. 6, no. 1, pp. 25 - 34.

Notes: a revised and expanded version of a conference paper presented at the 3rd Asia-Pacific Forum on Engineering & Technology Education, Changhua, Taiwan, 8 - 11 July 2001.

Abstract: Engineering education provides a thorough and systematic training in the design, development, maintenance and management of complex technical systems. While such education provides the necessary technical depth to graduates, many technical systems are best understood

from the perspective of human and socio-economic relationships. A case in point may be Fred Brooks' law that states adding more developers to a late software engineering project will only make it even more behind schedule. Brooks' law is based on the understanding that additional, new software engineering staff will need time to come up to speed with the project and in doing so will divert the existing developers from their primary tasks. While Brooks' law is intuitively appealing, students and practicing software engineers really have no way of testing its efficacy in their particular situations. A tool to overcome this difficulty may be system dynamics. System dynamics is a systems thinking methodology for building quantitative and qualitative models of complex situations so that they can ultimately be better understood and managed. Accordingly, it can be argued, that system dynamics should be an essential part of the education of engineers from most, if not all, of the major disciplines.

Caulfield, C. W., Kohli, G. and Maj, S. P. (2004). 'Sociology in Software Engineering.' *Proceedings of the 2004 American Society for Engineering Education Annual Conference & Exposition* (Salt Lake City). American Society for Engineering Education.

Notes: ERA category C.

Abstract: The sociology of software project management is an often under-represented component in the education and professional development of software engineers even though factors such as team formation, role assignment, motivation, training, hiring, and many other peopleware practices have been identified many times as at least equally important to the success of software projects as the technical. The reasons for this may be two-fold: the seeming arbitrariness of the sociological factors in software development is at odds with the formal and familiar technical aspects; and the lack of suitable tools with which to model and understand human dynamics.

However, these impediments may be overcome. For example, system dynamics is a modelling approach to dynamic socio-technical problems, stemming from the work of Forrester at MIT and since developed, that allows a modeller to mix soft variables (morale, perceptions, motivations) with familiar hard variables (time, cost, resources). A system dynamics

model is not so much a tool for time-point prediction, but more of an experimental device to see how certain variables might change over time under the influence of unappreciated causal relationships, dynamic complexity, and structural delays. The end result is hopefully a more informed mind set with which to manage the situation at hand.

By way of illustration, this paper presents some initial results of a system dynamics model based on Frederick Brooks' well-known informal law which warns against adding more software developers to a late project for risk of making matters worse. Brooks' law, the crystallisation of many years of practical software project experience, has been critiqued many times in the literature and generally enjoys wide support, making it a solid basis for any model of the socio-technical aspects of software project management. However, it operates at a high level of aggregation and is most often associated with large-scale software development projects. In contrast, the system dynamics model presented here creates a small-team, small-project environment more likely to be encountered by software engineers in the current market.

Caulfield, C. W. and Maj, S. P. (2007). 'Come Play.' In M. Iskander (ed.) *Innovative Techniques in Instruction Technology, E-Learning, E-Assessment, and Education*, pp. 86 – 91. Springer. doi: 10.1007/978-1-4020-8739-4_15.

Abstract: Games have been used as learning tools in many different business, military and social environments, but appear to be under-represented in a critical modern situation—software engineering: the systematic, disciplined, and quantifiable approach to the development, operation, and maintenance of software. Despite the name, software engineering may not enjoy the same standing as the more established engineering professions. Anecdotal evidence suggests that an urgent software crisis exists (a gap between expectations of software and the product and performance actually delivered) and has been growing since the 1960s. While quantitative data proving the existence of a software crisis is thin, it might be conceded that software engineering has room for improvement. This paper presents a case that the use of games as a research tool in software engineering needs to be more fully explored and

an opportunity exists to use games to tackle some of the current issues in the field.

Caulfield, C., Veal, D. and Maj, S. P. (2011). 'Teaching Software Engineering Project Management – A Novel Approach for Software Engineering Programs.' *Modern Applied Science*, vol. 5, no. 5 (October), pp. 87 – 104.

Notes: ERA category A.

Abstract: In response to real and perceived short-comings in the quality and productivity of software engineering practices and projects, professionally-endorsed graduate and post-graduate curriculum guides have been developed to meet technical developments and evolving industry demands. Each of these curriculum guidelines identifies better software project management skills as critical for all graduating students, but they provide little guidance on how to achieve this. One possible way is to use a serious game — a game designed to teach and educate players about some of the dynamic complexities of the field in a safe and inexpensive environment. This paper presents the results of a qualitative research project that used a simple game of a software project to see if and how games could contribute to better software project management education. Initial results suggest that suitably-designed games are able to teach software engineering and project management concepts at higher-order Bloom taxonomy levels.

Caulfield, C. W., Veal, D. and Maj, S. P. (2011). 'Implementing System Dynamics Models in Java.' *International Journal of Computer Science and Network Security* vol. 11, no. 7 (July), pp. 43 – 49.

Notes: ERA category C.

Abstract: For a research project into the value of serious games — games that teach and educate — in software engineering and project management education, a game called Simsoft was developed. Two keys parts of Simsoft were the system dynamics engine that captured the fundamental causal relationships of the software project being modelled; and the Java dashboard through which the players entered their project decisions. Java

also provided a means of saving the players individual decisions so these could later be analysed and replayed. While there are currently no Java libraries for implementing system dynamic models, a system dynamics model is simply a collection of non-linear differential equations, and open-source Java libraries for these do exist. Therefore, it is possible to implement a system dynamics model in Java and take advantage of the features of a powerful, general purpose programming language. This paper describes how the model behind Simsoft was created using system dynamics modeling tool called iThink and how the model was subsequently implemented in Java using the Apache Commons Mathematics library.

Caulfield, C. W., Veal, D. and Maj, S. P. (2011). 'Teaching Software Engineering Management – Issues and Perspectives.' *International Journal of Computer Science and Network Security* vol. 11, no. 7 (July), pp. 50 – 54.

Notes: ERA category C.

Abstract: The ACM/IEEE regularly proposes guidelines for software engineering education, in particular what should be part of the software engineering core body of knowledge and how this knowledge can be taught. The 2004 curriculum guidelines define seven student outcomes, two of which relate to teamwork and project control, and one Software Engineering Education Knowledge (SEEK) area on software management. The software management knowledge area is concerned with the entire software development life cycle and hence the control of people and processes. Significantly, the majority of topics within this area are classified with the Bloom taxonomy level of Application i.e. ability to use learned material in new and concrete situations. However the laboratory and assignment exemplars fail to demonstrate the dynamic, human centered complexity of project management. Simsoft, a serious game, has been designed to potentially address this pedagogical gap.

Caulfield, C. W., Xia, J., Veal, D. and Maj, S. P. (2011). 'A Systematic Survey of Games Used for Software Engineering Education.' *Modern Applied Science.*, vol. 5, no. 6 (December), pp. 28 – 43.

Notes: ERA category A.

Abstract: Simsoft is a serious game— one that trains or educates— at the centre of a research project designed to see if and how games can contribute to better software engineering management education by helping software engineers and project managers explore some of the dynamic complexities of the field in a safe and inexpensive environment. A necessary precursor for this project was to establish what games already existed in the field and how effective they had been. To this end a systematic review of the literature was conducted using a collection of online science, engineering, education, and business databases looking for games or simulations used for educational or training purposes in software engineering or software project management across any of the SWEBOK knowledge areas. The initial search returned 243 results, which was filtered to 36 papers by applying some simple quality and relevance inclusion/exclusion criteria. These remaining papers were then analysed in more depth to see if and how they promoted education in the field of software engineering management. The results showed that games were mainly used in the SWEBOK knowledge areas of software engineering management and development processes, and most game activity was in Europe and the Americas. The results also showed that most games in the field have learning objectives pitched at the first rung of Bloom’s taxonomy (knowledge), most studies followed a non-experimental design, and many had very small sample sizes. This suggests that more rigorous research is needed into the efficacy of games in teaching software engineering management, but enough evidence exists to say that educators could include serious games in their courses as a useful and interesting supplement to other teaching methods.

Caulfield, C. W., Maj, S. P., Xia, J. and Veal, D. (2011). ‘Shall We Play a Game?’ *Modern Applied Science*. Vol.6, no. 1 (January), pp. 2 – 16

Notes: ERA category A.

Abstract: This paper presents the results of a qualitative research project that used a simple game of a software project to see if and how games could contribute to better software project management education, and, if

so, what features would make them most efficacious. The results suggest that while games are useful pedagogical tools and are well-received by players, they are not sufficient in themselves and must be supplemented by other learning devices.

A Case for System Dynamics

C W Caulfield, S P Maj

Computing Science, Edith Cowan University (ECU).

Perth, Western Australia.

ABSTRACT: Engineering education provides a thorough and systematic training in the design, development, maintenance and management of complex technical systems. While such education provides the necessary technical depth to graduates, many technical systems are best understood from the perspective of human and socio-economic relationships. A case in point may be Fred Brooks' law that states adding more developers to a late software engineering project will only make it even more behind schedule. Brooks' law is based on the understanding that additional, new software engineering staff will need time to come up to speed with the project and in doing so will divert the existing developers from their primary tasks. While Brooks' law is intuitively appealing, students and practicing software engineers really have no way of testing its efficacy in their particular situations. A tool to overcome this difficulty may be system dynamics. System dynamics is a systems thinking methodology for building quantitative and qualitative models of complex situations so that they can ultimately be better understood and managed. Accordingly, it can be argued, that system dynamics should be an essential part of the education of engineers from most, if not all, of the major disciplines.

INTRODUCTION

Engineering education provides a thorough and systematic training in the design, development, maintenance and management of complex technical systems. Without question such education provides the necessary technical depth to graduates. However, many technical systems are best understood from the perspective of human perceptions and also that of a wider socio-economic context. It is well documented that the success of technical projects, in many instances, is almost entirely dependent on these factors.

It is a curious paradox that the software industry has helped provide the means by which others have been able to automate, reengineer, and economy-scale their businesses, that is, reduce the human variable, and yet remains itself very people sensitive and intensive. For example:

Highly skilled people with appropriate experience, talent, and training are key to producing software that satisfies user needs on time and within budget. The right people with insufficient tools, languages, and process will succeed. The wrong people (or the right people with insufficient training or experience) with appropriate tools, languages, and process will probably fail. [4, p. 150]

Tom DeMarco, co-author of the often-cited *Peopleware* [6] has found that most software development managers agree with this premise that a project's sociology will contribute more to the final outcome than the project's technology. Sociology, in this context, means addressing issues such as team formation and dynamics, role assignment, hiring, motivation, workplace design, training, and many other peopleware practices. However, the same managers do not conduct their projects with this regard, and instead focus on that aspect they are most comfortable with: technology. *"The evident reason for this is that the manager knows how to do technology, but not how to*

do sociology. He/she doesn't know how to manage" [22, p. 149].

One of the golden rules of software engineering texts maybe a case in point— Fred Brooks' [3] informal law that states that adding more software developers to a late project will only make it later. Brooks' law is based on the understanding that the new developers will need time to come up to speed with the project and in doing so will divert the existing developers from their primary, and now critical, tasks. While Brooks' law is intuitively appealing, students and practicing software engineers really have no way of testing its efficacy in their particular situations because such systems are difficult to model.

One possible way to address such situations is by using the systems thinking methodology, system dynamics.

System dynamics is concerned with building quantitative and qualitative models of complex problem situations and then experimenting with and studying the behaviour of these models over time. Often such models will demonstrate how unappreciated causal relationships, dynamic complexity, and structural delays may lead to counter-intuitive outcomes of less-informed efforts to improve the situation. System dynamic models make room for soft factors such as motivation and perceptions so that engineering projects can ultimately be better understood and managed.

This paper presents some initial results of implementing a simple model of Brooks' law using a system dynamics modelling software package called iThink to support the argument that system dynamics should be an essential part of the education of engineers from most, if not all, of the major disciplines.

SYSTEM DYNAMICS

In the late 1950s, Jay Forrester of the Sloan School of Management at MIT was asked by General Electric to review the operations of their Kentucky appliance parts plant. The company was concerned about the oscillating nature of their production cycles that often saw periods of intense activity followed by times of virtual dormancy during which workers had to be laid off. Fluctuating demand and normal business cycles did not seem to adequately explain the situation. Coming from an electrical engineering background, and with a keen interest in management science, Forrester approached the problem systematically, but with just a pencil and a note pad. Starting with columns for inventory, employees and orders, and factoring in “the policies they were following, one could decide how many people would be hired in the following week. This gave a new condition of employment, inventories, and production” [23]. Forrester’s calculations amounted to a simulation of the system operating at General Electric’s plant.

Stemming from this first analysis came an article for the *Harvard Business Review* in 1958 entitled “Industrial Dynamics— A Major Breakthrough for Decision Makers” with the theme being developed and expanded in the seminal work, *Industrial Dynamics* [7]. Industrial dynamics became system dynamics, reflecting its use in areas other than business and industry.

For some time following the publication of *Industrial Dynamics*, system dynamics was used as a tool for looking at big-picture issues such as urban decay, major sociological conditions, and world economics [8, 9, 11]. But in more recent times, system dynamics has come back from the big end of town and has been finding a purpose for itself in a range of business and social applications. Instrumental in this change have been the publication of Peter Senge’s *The Fifth Discipline* [13], and the development of intuitive, graphical software packages which have made system dynamic modelling more democratic by hiding the computer source-code look of traditional models. As a measure of this democracy, system dynamics now finds a place for itself in primary and secondary schools in the United States, Australia, and Europe well beyond its ground zero at MIT.

To more formally define system dynamics we might say that it:

is concerned with creating models or representations of real world systems of all kinds and studying their dynamics (or behaviour). In particular, it is concerned with improving (controlling) problematic system behaviour... The purpose in applying System Dynamics is to facilitate understanding of the relationship between the behaviour of the system over time and its underlying structure and strategies/policies/decision rules. [16, p. 2]

A key element of this definition is the need to build a model of the system under consideration. The model is used to help understand the patterns of change, or dynamics, that a system exhibits over time and to identify the conditions which cause these patterns to be stable or unstable. This knowledge of the system can then suggest what kinds of prescriptions and approaches to governing it will work and what kinds will not [14, p. 248].

However, building system dynamics models demands patience and thought. Translating real-world information into model elements is still an inexact science— trial and error can be just

as valid as considered judgment based on experience. Perhaps a useful parallel can be drawn with that other hard, inexact activity: finding object-oriented classes. Bjarne Stroustrup, the creator of C++, notes that in design and programming there are no cookbook methods that can replace intelligence, experience and good taste; *even he just tries things* [15, p. 362]. The lesson for system dynamics modellers would seem to be the same: just start, try things, take advice of experienced modellers, and then iterate, iterate, iterate.

Yet, the effort of building a system dynamic model has some benefits:

- Modelling brings about an understanding of the system because of the analytical and critical thinking process it calls for. It helps bring to the surface the mental models driving the current situation— those models “*that one carries around in one’s head for dealing with a problem or situation. Such a model maybe based on experience or intuition, or on folklore and myth; it may be influenced by politics and a wide spectrum of human emotions*” [17, p. 86]. Mental models may also be totally inappropriate or counter-productive, or equally priceless, but unless we turn them into something more tangible, we will never know.
- System dynamics models allow room for both quantitative or hard variables, being things we can measure directly like program size, staffing numbers, dollars spent; and qualitative or soft variables such as motivation, commitment, confidence, or perceptions. Soft variables have traditionally been left out of engineering models because they are difficult to measure and their importance may have been under-estimated. Yet, “*if you omit ‘soft’ variables you run the risk of failing to capture something essential to driving human affairs. Leaving out something so essential is the only hypothesis that you can reject with absolute certainty!*” [21, p. 9-1). A system dynamics model can therefore be more informed about its problem space.

With a system dynamics model in hand and George Box’s tongue-in-cheek caution in mind (all models are wrong, but some are useful), the model can be to run. Certain variables can be held steady while others are changed, it can be placed under stress, and tested for sensitivities and leverage points. In short, the model can be experimented with to better understand the present situation and to search for alternatives for improvement. “*The alternatives may come from intuitive insights generated during the [initial analysis], from experience of the analyst, from proposals advanced by people in the operating system [or, in the] experience, art, and skill for imagining the most creative and powerful policy alternatives*” [20, p. 246].

Peter Senge points out that the causes of many problems “*lay in the very well-intentioned policies designed to alleviate them. These problems were actually ‘systems’ that lured policy makers into interventions that focused on obvious symptoms not underlying causes, which produced short-term benefit but long term malaise, and fostered the need for still more symptomatic interventions*” [13, pp. 14 – 15].

By simulating a problem space using a system dynamics model it is possible to potentially make more informed decisions

about events beyond our bounded rationality safe from the dangers of real-world experimentation.

BROOKS' LAW

During the 1950s and early 1960s, Fred Brooks worked for IBM as a programmer and hardware architect. In 1964 he became the manager of IBM's Operating System/360 development, a large-scale and complex project intended to provide IBM's mainframe computers with a leading-edge operating system. To give an idea of the size of the project "*the initial Windows NT project required about 1,500 staff-years of effort, but the development of IBM's OS/360, which was completed in 1966, required more than three times as much effort*" [10, p. 4].

His experiences, frustrations, and joys during this time, and his observations of the wider industry after moving to the University of North Carolina, are embodied in the collection of essays *The Mythical Man-Month* [3]. The title refers to that fundamental unit of measurement and scheduling, the man-month; a unit that Brooks believes is often misunderstood:

Cost does indeed vary as the product of the number of men and the number of months. Progress does not. Hence the man-month as a unit for measuring the size of a job is a dangerous and deceptive myth. It implies that men and months are interchangeable. [3, p. 16]

His law that states adding more software developers to an already late project will only make the problem worse is based on this lack of interchangeability of manpower and time. The cause lies in two areas:

- The new developers will need to be acquainted with the overall aims of the project, its strategy and the general plan of work. During this time the new developers will not be full contributors and will likely divert the existing developers away from their primary tasks.
- If a group of developers, n , need to coordinate their efforts with each other then the number of communication paths can be represented by $n(n - 1)/2$. This represents an interaction overhead, which may be realised in the form of project meetings, technical walkthroughs, and complying with any progress reporting requirements.

Brooks' law is intuitively appealing and is generally supported in the literature [2, 5, 12, 17]. Writing in the anniversary edition of *The Mythical Man-Month* in 1995, Brooks acknowledged that his law was outrageously simplified yet he still felt that it was the:

best zeroth-order approximation to the truth, a rule of thumb to warn managers against blindly making the instinctive fix to a late project.[3, p. 275].

Yet, turning Brooks' law into something more than a rule of thumb we should be able to test whether it is a useful concept outside the large-scale big business and government projects Brooks' was most familiar with.

MODEL EXPLANATION

The following model of Brooks' law has been created using a system dynamics modelling package called iThink. The grammar of iThink consists of only four basic elements (stocks, flows, rates, and connectors) and is largely intuitive so it won't be expanded upon here. Further details are provided in the appendix.

In addition, a range of assumptions is made that will naturally vary according to local conditions. What is important is not so much the magnitude of these assumptions in this particular instance, but that they are relevant to the problem space under consideration and that they can be changed as needed.

Looking to the model, we have a hypothetical software development project in hand that has been estimated at 36-man months, or 6240 hours, and must be completed within six months. To meet this deadline a staffing level of six developers has been approved. However, the project starts with only five developers, three of whom are experienced, meaning they are aware of the objectives of the project and the plan of work; and two who are new-hires. It is assumed that the new-hires will only be half as productive as their colleagues, but will gradually come up to speed as they are assimilated. This transitioning from new-hires to experienced developers has been set at three months.

Recruiting is under way to bring the team up to full strength but advertising the position, assessing the applicants, and making a decision all takes time. Therefore, a delay of some two months is not unreasonable [17, p. 98]. At the same time staff are likely to leave. For the purposes of this model, it is assumed that the average employment time will be nine months, and for simplicity, it is assumed that developers will not quit the team before becoming experienced developers.

Figure 1 represents to model to this stage.

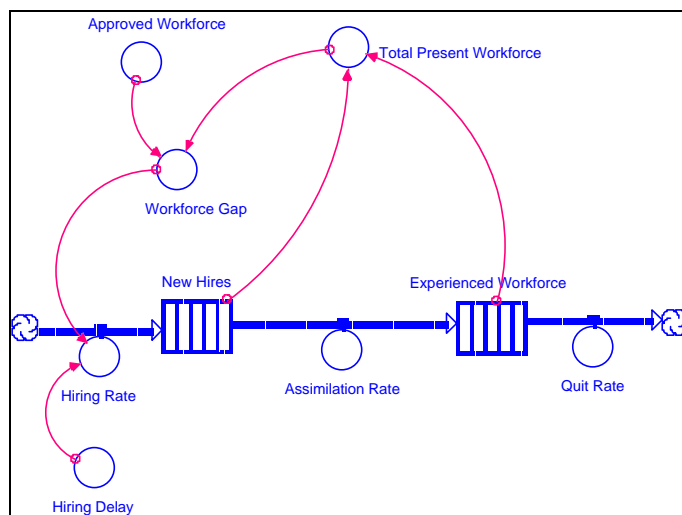


Figure 1

Staff enter the 'plumbing' of the iThink diagram from the left, progressing to the right as they pass from being new-hires to experienced developers until they perhaps eventually leave the team. The *Total Present Workforce* will therefore be the sum of the two groups of developers. If the *Total Present Workforce* is less than the *Approved Workforce*, a *Workforce Gap* will exist and the hiring process will be initiated, subject to the prescribed delay of two months.

Figure 2 represents the workflow of the project.

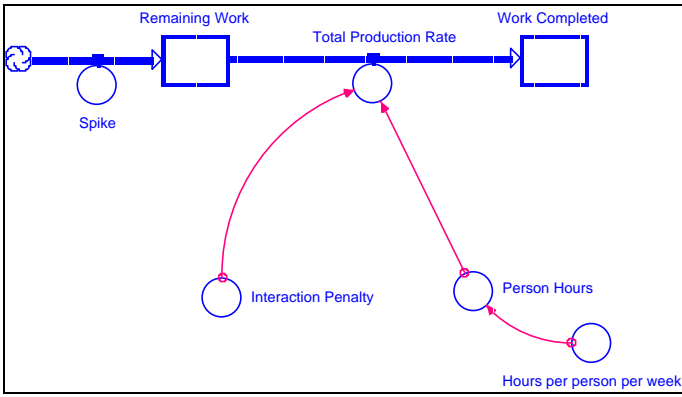


Figure 2

The team has 36 man-months of work to complete, therefore at the start of simulation *Remaining Work* will represent this amount. Work units will flow towards *Work Completed* at a rate determined by the overall productivity of the team. Occasionally, there may be a spike in *Work Remaining* if the scope of the project is expanded or if the original work estimates have been found to be underestimated.

The total productivity of the team will be a function of the total workforce, the number of hours each person works per week, which has been set at a standard 40, the assumed productivity of the new-hires versus their more experienced colleagues, and taking into account the interaction overhead required to coordinate all the individual development efforts. For the purposes of this model, it is assumed that the interaction overhead represents one hour per developer per week per communications path. If there are five developers, this equates to ten communications paths, and therefore ten hours per week per developer consumed in this overhead.

The model in its entirety is represented by figure 3.

MODEL RESULTS

Setting the model to run under the initial conditions described above produces the graph in figure 4.

The approved workforce is six developers, but at the start of the project only five are on hand. After allowing for the recruiting delay, the number of new-hires increases reflecting the addition of one extra developer. And, over time, the number of experienced workers increases as the new-hires come up to speed with the project. After nine months, or 36 weeks, experienced developers begin to leave, which initiates the hiring process again.

Even allowing for the fact that the project started with one developer less than required, the graph indicates that simply dividing the effort by the number of staff on hand will not

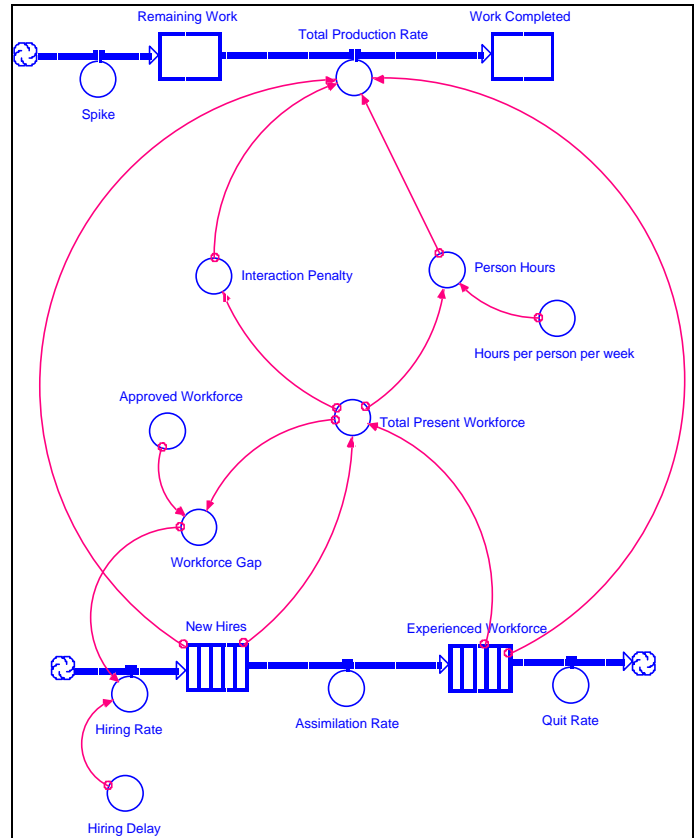


Figure 3

yield an overall completion time. With the best will, the project will take nearly twelve months to complete rather than the original six.

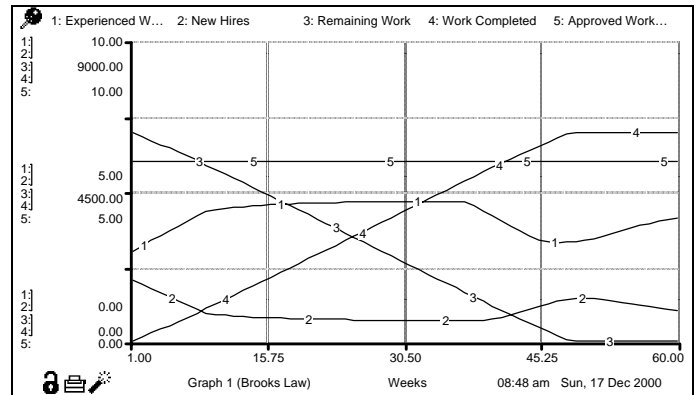


Figure 4

Assume now that the project has been underway for five months, or 20 weeks, when it is discovered the original man-month estimates were understated.



Figure 5

Another twelve man-months of work have been assessed. Assuming 40-hour weeks and a present staff of six developers, this means the project will be extended by another eight weeks. To bring this figure down, the project manager decides to increase the approved staffing to eight developers. The resulting graph under these circumstances now looks figure 5.

Despite bringing on more staff, the project is still not able to hit its revised completion date and now takes nearly eighteen months to complete.

CONCLUSIONS

The system dynamics model of Brooks' law presented here is necessarily generic and simplified, and is part of an as yet incomplete project. But, even at this level it is one realisation of a mental model that can now be shared, discussed and hopefully improved upon.

The results in this case tend to support Brooks' law that adding more software developers to an already late project will only make matters worse. However, this may not always be so. For example, using a more detailed model of Brooks' law, Abdel-Hamid and Madnick [1, 18, 19] found that if the developers are added early in the project rather than towards the end, the project will have more chance of hitting its deadlines. But, without the model, the belief that this might be so would have been without support.

Making system dynamics a part of all engineering disciplines would seem to be an incremental rather than a discontinuous step since engineers are likely already familiar with the benefits of building models. System dynamics can bring to this process its underlying theme that more informed socio-technical models are possible.

As a means of capturing mental models, building decision flight-simulators, and communicating complex ideas at a higher level than verbal descriptions, system dynamics deserves serious consideration. But, in response, the methodology demands the patience to understand its concepts, nuances, and power.

REFERENCES

[1] Abdel-Hamid, T. K. and Madnick, S. E., *Software Project Dynamics: An Integrated Approach*. Englewood Cliffs: Prentice Hall (1991).

[2] Boehm, B., *Software Engineering Economics*. Upper Saddle River: Prentice Hall (1981).

[3] Brooks F. P., *The Mythical Man-Month: Essays on Software Engineering*, anniversary edition. Sydney: Addison-Wesley (1995).

[4] Davis A. M., *201 Principles of Software Development*. Sydney: McGraw-Hill (1995).

[5] DeMarco, T., *The Deadline: A Novel About Project Management*. New York: Dorset House (1997).

[6] DeMarco T. and Lister T., *Peopleware: Productive Projects and Teams*, second edition. New York: Dorset House (1999).

[7] Forrester J. W., *Industrial Dynamics*, Waltham: Pegasus Communications (1961).

[8] Forrester J. W., *Urban Dynamics*. Portland: Productivity Press (1969).

[9] Forrester J. W., *World Dynamics*. Portland: Productivity Press (1971).

[10] McConnell S., *After the Gold Rush*. Redmond: Microsoft Press (1999).

[11] Meadows D. H., Meadows D. L., Randers J., and Behrens W. W., *The Limits to Growth*. New York: Universe Books (1972).

[12] Pressman, R. G., *Software Engineering: A Practitioner's Approach*, fourth edition. New York: McGraw-Hill (1997).

[13] Senge P. M., *The Fifth Discipline: The Art and Practice of the Learning Organization*. Sydney: Random House (1990).

[14] Stacey, R. D., *Strategic Management & Organisational Dynamics*. Melbourne: Pitman Publishing (1996).

[15] Stroustrup B., *The C++ Programming Language*, second edition. Sydney: Addison-Wesley Publishing Company (1993).

[16] Wolstenholme E. F., *System Enquiry: A System Dynamics Approach*. Brisbane: John Wiley & Sons, (1990).

[17] Yourdon E., *Rise and Resurrection of the American Programmer*. Upper Saddle River: Prentice Hall (1998).

[18] Abdel-Hamid, T. K., The Dynamics of Software Project Staffing: A System Dynamics Based Simulation Approach. *IEEE Transactions on Software Engineering*, 15, 2, 308 - 318 (1989).

[19] Abdel-Hamid, T. K. and Madnick, S. E., Lessons Learned from Modeling the Dynamics of Software Development. *Communications of the ACM*, 32, 12, 1426 - 1455 (1989).

[20] Forrester, J. W., System Dynamics, Systems Thinking and Soft OR. *System Dynamics Review*, 10, 2-3, 245 - 256 (1994).



[21] Richmond, B., Modelling "Soft" Variables. *An Introduction to Systems Thinking*, Hanover, High Performance Systems, 9-1 - 9-10 (1999).

[22] DeMarco, T., Non-Technological Issues in Software Engineering. *Proceedings of the 13th International Conference on Software Engineering*. Austin, USA, 149 - 150 (1991).



[23] Forrester J. W., The Beginnings of Systems Dynamics' [on-line]. Available WWW: <http://sysdyn.mit.edu/sd-intro/home.html> (1989).

APPENDIX I: THE LANGUAGE OF iTHINK

At its core, iThink is a language that can be used to tell a story. Therefore, system dynamic models described by it use the following elements of grammar to tell their story.

- Stocks, , are the nouns of iThink. They represent an accumulation of something at a point in time. The slatted stocks used in the above model are a special version known as conveyors. They work in the same way as normal stocks except that anything entering the conveyor 'rides' along it for a set period of time and then leaves.
- Flows, , are the verbs of iThink. Stuff flows through the pipe of the flow in the direction of the arrow and at a rate determined by the flow regulator in the middle. The flow regulator is fitted with a spigot that can

be conceptually tightened or loosened by other variables within the model. The cloud at the end of the flow represents the boundary of the model.

- Converters, , can be thought of as adverbs which modify flows. They are often used to break out the detail of the logic that might otherwise be buried within a flow and might be used to represent constant values. These typically influence the behaviour of the regulators on the flows
- Connectors, , tie the other three building blocks together. They represent inputs and outputs, not inflows and outflows. Connectors do not take on numerical values— they merely transmit values taken on by other building blocks.

A CASE FOR SYSTEMS THINKING AND SYSTEM DYNAMICS

CW CAULFIELD and SP MAJ

Department of Computer Science, Edith Cowan University, Perth Western Australia

1 Introduction

The title of this paper is too brief to be quite accurate. Perhaps with the following subtitle it does not promise too much: a review of systems thinking that considers its unique history and influences, paradigms and methodologies, and presenting a case for the system dynamics methodology as the best tool for the most diverse range of problem situations.

Systems thinking is a way of thinking that focuses on the relationships between the parts forming a purposeful whole. Its intellectual integrity draws from a number of fields and influences including philosophy, sociology, organisational theory, feedback thought, and a reaction against the method of science. Aspects of these influences have been examined.

Systems thinking can be practiced in more than one way. A collection of methodologies representative of both sides of the main hard/soft divide within the paradigm have been evaluated including soft systems methodology, systems engineering and analysis, operations research, organisational cybernetics, interactive planning, and organisational learning. Each has been considered in terms of its advantages and disadvantages and most appropriate applications.

Completing the list of system methodologies is a special case in the instance of this paper— system dynamics. System dynamics is concerned with building computer models of complex problem situations and then experimenting with and studying the behaviour of these models over time. Often such models will demonstrate how unappreciated causal relationships, dynamic complexity, and structural delays may lead to the counter-intuitive outcomes of less-informed efforts to improve the situation. System dynamic models make room for soft factors such as motivation and perceptions so that problem spaces can ultimately be better understood and managed.

A case is made as to why systems thinking in general and system dynamics in particular represent a choice of first resort for the broadest range of problem spaces. In brief, the argument is they boast the best tool set, they have the best intellectual credentials, and they are best suited to contemporary business and social situations.

2 Systems Thinking History and Influences

Humans have always been a part of systems but for the most part there was no realisation of the actuality of systems. Primitive societies accepted their role in a divinely given order of things without too much contemplation, and adjusted themselves as circumstances required. With industrialisation, political, economic and social systems became more noticeable but no more easy to grasp. “A search for orderly structure, for cause and effect relationships, and for a theory to explain system behaviour gave way at times to a belief in random, irrational events” [6, p. 1-1].

However, philosophers and sociologists have attempted some explorations.

In the early nineteenth century, the German Idealist philosopher Georg Hegel (1770 – 1831) conceived of an enormously broad, holistic fashion of thinking in which there was room for everything— logical, natural, human, and divine. Hegel believed that the truth about reality could not be grasped by studying phenomena in isolation; rather, a higher, more abstract philosophical vantage point was needed.

Although likely unappreciated and unintended at the time, Hegel’s dialectic also contains a key systemic construct— a negative feedback loop. The tension between thesis and antithesis, between the desired and the actual, eventually forces a new state of affairs, the synthesis [20, p. 71].

Writing around the turn of the last century, the French sociologist Emile Durkheim (1858 – 1917) carefully and critically absorbed the ideas of the French sociologist Auguste Comte (1798 – 1857) and other contemporaries such as Herbert Spencer (1820 – 1903), particularly accepting the notion that the scientific viewpoint was the best from which to study social reality. However, Durkheim did not believe that scientific reductionism or “an analysis of the parts which existed in the social organism and the role they performed was adequate as an end of sociological analysis” [1, p. 44]. Instead, he felt that causal analysis (why) of social phenomena was required in addition to functional analysis (what). For example, the study of a social formation needs to

take account of the social and historical forces that bring it into being and allow it to operate. Any such group, though not necessarily superior to its individual parts, is different from them and demands an explanation on the level peculiar to it. That is, the whole is more than just the sum of its parts [9, pp. 22 – 26].

If a common thread can be said to run through the work of this small collection of social theorists, it may be that each saw value in using biological and mechanical metaphors to understand social phenomenon. At a period in European history dominated by industrialisation, positivism, and evolutionism, it seems only logical that similar threads became woven into their writings. Even though the overly scientific and rigorous methods they advocated are at some odds with softer, current versions, the underlying systemic understanding shows itself as an idea of some age and magnetism.

In more modern times organisational theorists have also contributed to the field of systems thinking particularly through open systems theory: a way of thinking that recognises the dynamic interaction of the system with its environment in which inputs are transformed by some internal process and made into outputs.

Influential in early open systems theory was the US sociologist and Harvard professor, Talcott Parsons (1902 – 1979). He advocated a structural-functionalist approach to analysing social systems, an approach built upon the biological metaphor and that focuses on the concepts of holism, interrelationships between parts, structure, functions, and needs [1, p. 50].

Parson's writings have been criticised as being too conservative and avoiding or being unable to explain change and dysfunction in social systems [11]. More able to do this was a contemporary of Parsons, Robert Merton, who believed that the structural-functionalist approach was valuable because it required the viewer to examine the consequences of social action, that is, its latent functions, rather than relying solely on superficial manifest functions. Even so, less fully developed in Merton's theory was an explanation of why dysfunctions might continue. It may be that Merton had not stepped back far enough to see these dysfunctions as ongoing issues, particularly if he accepted Vilfredo Pareto's (1848 – 1923) equilibrium proposition:

His view of society was that of a system of interrelated parts which, though in a continual state of surface flux, were also in a state of underlying equilibrium, in that movements away from the equilibrium position were counterbalanced by changes tending to restore it. [1, p. 47]

That is, deviations from the norm are mended by the system. The feedback theory underlying Pareto's model of society is premised on the mechanical, rather than the biological, metaphor and herein may lay a reason why dysfunctions continue in spite of a Pareto system's innate search for equilibrium. The mechanical metaphor assumes that any deviation from the norm will feed back into the system and be invariably acted upon by certain rules. Yet, in any system composed of decidedly unmechanical humans this feedback may be indeed be handled in this way, or it may just as likely be misinterpreted or arbitrarily ignored.

To more formally define the feedback that Pareto talks of we might say that it is a process through which an action (an event or piece of information) passes through a series of causal relationships to eventually affect the original action.

Examples of virtually fully developed concepts of feedback thought can be found in the inventions and writings of the ancient Greeks while many of the most influential machines of the Industrial Revolution employed some form of automatic regulation [17].

It is interesting to note that after a long hiatus, there was a sudden explosion of feedback inventions in Europe at the time of the Industrial Revolution. Mayr [17] believes that technical and economic factors alone do not adequately explain this sudden burst of interest in automatic regulation. In fact, the same interest had a much wider cast as the writings of some of the philosophers and sociologists discussed already demonstrate. It would seem that at a point in time marked by great social, economic, and political uncertainty, largely brought about by fundamental technological changes, people at all levels were searching for meaningful stability and structure.

Given that feedback thought has a history of many centuries and was being used intuitively and elegantly, if unknowingly, in many fields, it is perhaps surprising that its self-awareness is only relatively recent. Richardson [20] believes that Rosenblueth, Wiener, and Bigelow's 'Behavior, Purpose, and Teleology' (1943) was the first published work to link human systems with the engineer's concept of feedback. In this article, the authors make the distinction between non-purposeful behaviour, which is basically random, and purposeful behaviour, which is directed towards some goal. If signals from the goal modify the action in the course of the behaviour, then feedback is happening.

In *Cybernetics, or Control and Communication in the Animal and the Machine* (1948), Norbert Wiener expanded on the theme, in the process coining the word cybernetics, being a metaphorical application of the Greek *kubernetes*, meaning steersmanship. Wiener and his colleagues had applied

the concept during World War II looking for ways to develop and refine devices for the control of gunfire.

Traces of holistic thinking can therefore be found in many areas of study. Yet from an early stage each discipline had been using holistic thinking to cope with its own elements of complexity and had tended to use a language unique to its environment, meaning that the systems movement was late in gaining a degree of self-awareness. It was not until the late 1940s that the organismic biologist Ludwig von Bertalanffy appreciated that the parallel ideas in various disciplines could be generalized in a systems theory.

As a biologist von Bertalanffy was interested in the nature of life but noted that an organism's constituent physico-chemical processes did not explain all there was to know. Never a vitalist, von Bertalanffy suggested that a return to the organismic biology that preceded the invention of the microscope was a more fruitful avenue of thought. That is, organisms should be studied as irreducible, whole systems, contrary to a central tenet of the method of science that advocated reductionism.

From the 1950s, von Bertalanffy shifted his focus from the biological sciences to the methodology of science. He was concerned that scientific endeavour was following too faithfully one of its own rules:

Modern science is characterized by its ever-increasing specialization, necessitated by the enormous amount of data, the complexity of techniques and of theoretical structures within each field. Thus science is split into innumerable disciplines continually generating new subdisciplines. In consequence, the physicist, the biologist, the psychologist and the social scientist are, so to speak, encapsulated in their private universes, and it is difficult to get word from one cocoon to the other. [23, p. 30]

Despite this fragmentation, von Bertalanffy noticed that there existed a certain parallelism of general cognitive principles in fields such as chemistry, physics, biology, and sociology, made all the more striking by having developed independently in each [23, p. 31]. If this underlying isomorphism could be captured and made known then a tool would be at hand to reunify science and to move it forward more quickly. With the publication of two influential articles in 1950, 'The Theory of Open Systems in Physics and Biology' and 'An Outline of General Systems Theory', von Bertalanffy introduced the tool he had conceived for the task— general systems theory (GST).

However, the generality of an analytical framework such as GST is both a weakness and a strength:

- Weakness: by taking a holistic view, general systems theory takes away the comfort of mastering details and means understanding relationships instead of absolute facts. However, the relatively vague, initial totality is transitory. As a general understanding of the overall system is attained, the focus of study can then narrow to the analysis of details, but with a broader understanding in mind.
- Strengths: if we study the parts of a system alone, we will lack essential knowledge of the whole; and if we study the overall entity without comprehending its makeup, we will lack a fundamental awareness. General systems theory is a coherent way of resolving the parts-versus-whole dilemma.

Being aware that the word 'paradigm' can be easily misused, GST could be called a paradigm shift. According to Kuhn [15] paradigm shifts occur when the prevailing normal science is unable to answer those questions left in the too-hard basket. The reductionist method of science had certainly not dealt adequately with all the difficult problems it had been presented with, but then neither has systems theory. The reason lies in each, in their purest paradigmatic form, being suited to particular tasks. This theme of selecting the right tool for the job at hand recurs when we come to consider specific ways of practicing systems thinking.

3 The Systems Thinking Paradigm and Methodologies

The systems community is no more immune to paradigm or methodological racism than any other. In fact, Midgley [18] talks of paradigmatic wars and caustic sniping between the different schools of system thought, with the two dominant combatants being hard and soft systems thinking. The literature generally supports the distinction between the two on the basis of their most-suited problem contexts:

- Hard systems thinking is best applied to well-defined, goal-oriented, quantifiable, and real-world problems. Examples would include systems analysis and engineering and old-style operations research
- Soft systems thinking is best applied to ill-defined, fuzzy problem spaces, usually made this way because of the unpredictability of people, uncertainty, and other cultural considerations. Examples would include soft systems methodology and soft operations research.

Hard systems thinking predates its soft relation and retains traces of its origins in World War II logistical and scientific support of military operations.

In peacetime the paradigm found purpose in government and industry.

But in less predictable times, hard systems thinking was found wanting when it was applied to problems a good deal softer than its 'home' disciplines of engineering and defense economics, mainly because precise objectives were not so easy to pin down [2, p. 141]. Something else was needed to analyse softer, ill-defined problems.

Enter soft systems thinking.

Before soft systems thinking had properly settled itself, however, its methodological and epistemological foundations were being challenged. Around 1990, two main areas of concern had arisen:

- "That the interpretive theory underpinning soft systems thinking is inadequate for understanding and acting in social situations where there are inequalities in power and economic relations" [4, p. 79].
- That soft systems thinking practised too rigorously paradigm incommensurability, refusing to accept that any of the tenets of hard systems thinking might have value.

Enter, this time, critical systems thinking, a research perspective embracing three fundamental commitments: critical awareness, emancipation, and methodological pluralism.

In essence, critical systems thinking argues that practitioners be just that—critical. It accepts that no single paradigm or methodology is best in all circumstances and that an informed judgment needs to be made based primarily on the nature of the problem space being addressed.

In this light, a literature review of a representative range of systems thinking methodologies has been conducted. The methodologies include soft systems methodology, operations research, organisational cybernetics, interactive planning, organisational learning, systems analysis, systems engineering, and system dynamics. Each was critiqued from a critical systems thinking viewpoint of selecting the most appropriate methodology for the issue at hand.

However, not all authors accept that, when faced with a particular problem, we are free to choose an appropriate methodology from within a certain paradigm: "paradigms cannot be like spectacles that we can change when necessary" [19, p. 452].

If we take the critical systems thinking view that methodological pluralism is an attainable concept, then a valid question to ask at this point is which is most appropriate in certain circumstances? Research since the early 1990s at the University of Hull in the United Kingdom has been directed at this question. Using the principles of critical systems thinking as a basis, total systems intervention (TSI) is a meta-methodology that:

uses a range of systems "metaphors" to encourage creative thinking about organisations and the difficult issues their managers have to confront. These metaphors are linked by a framework (a "system of systems methodologies") to various systems approaches, so that once agreement is reached about which metaphors are most relevant to an organisation's concerns and problems, an appropriate systems-based intervention methodology (or set of methodologies) can be employed. Choice of an appropriate systems methodology will guide problem management in a way that ensures that it addresses what are found to be the main concerns of the particular organisation involved [5, p. 322].

The system of system methodologies is typically that proposed by Jackson and Keys [14]. The authors define a matrix made up of the two essential dimensions of any problem space: the nature of the people who are the would-be problem solvers, described using the language of industrial relations; and the environment or context of the problem.

The value of Jackson and Keys matrix is that it "helps get inside methodologies and to assess the fundamental assumptions that they hold about the nature of social reality" [3, p. 129] so that the best tool for the job at hand can be used. For example, if the problem context is seen to be one in which there are differing opinions that might still allow consensus (pluralist), and none of the participants seem to have the whole picture (systemic), then a methodology based on systemic-pluralist assumptions is the most appropriate, for example soft systems methodology or interactive planning.

However, Jackson warns those using the system of systems methodologies to be critically aware of their particular choice since "the aim is... [also] to reveal the particular strengths and weaknesses of available systems approaches and to make explicit the consequences, because of the assumptions each makes about systems and the relationships between participants, of using any of these" [13, p. 664]. That is, the system of systems methodologies should not be used slavishly.

As meta concepts, critical systems thinking and total systems intervention have been criticised for following too closely the functionalist's predilection for classifying things like 'insects on pins in shirt boxes'. If we take this criticism to an absurd end then we might not classify or organise anything. Therefore, in reviewing the collection of systems methodologies here, a more productive line of thought has always been held: at a time characterised by increasing detail and dynamic complexity, paradigm blindness is wasteful. Instead, problem solvers and thinkers need to be practised in the art of scanning for ideas—greedy almost in looking for

concepts, visions, tools or paradigms that make sense to them, at this time, and in their organisations.

4 Conclusions

Humans need help; help in coping with the information overload made possible by technology; help in dealing with the new dynamic complexities of the shift to knowledge economies; and help in compensating for those human attributes that often mean we do not act to our own best advantage.

The argument of this paper has been that systems thinking has the historical intellectual integrity and practical application to provide this help.

Systems thinking offers an opportunity to become more fully aware, to make informed decisions that extend beyond our otherwise bounded rationality, and to view problem spaces in their proper context. It does this by taking a worldview opposite to the atomised simplicity or specialised decomposition that Laszlo [16] criticises. Breaking a whole into its parts is analysis, through which we gain knowledge. Building parts into wholes is synthesis, through which we gain understanding. Through this understanding it becomes possible to achieve change that truly address the root causes of problems, rather than simply hoping that it might do so.

Systems thinking also fosters a collective understanding of a problem situation. Many of the tools of systems thinking, such as causal loop diagrams, rich pictures, or system archetypes, are visual rather than verbal descriptions. "A systems diagram is a powerful means of communication because it distils the essence of a problem into a format that can be easily remembered, yet is rich in implications and insights" [10, p. 6].

Yet, systems thinking is not as widely practised as these points might suggest it should be.

Systems thinking does not provide the linear quick fix needed in many political and organisational settings. In these situations, action, any action, is mistaken for achievement so that a problem deferred or shifted is a problem solved. Systems thinking forsakes the quick fix for hopefully the right fix.

Furthermore, the counter-intuitive and sometimes painful solutions offered by systems thinking can be hard to sell:

There are no utopias in social systems. There appear to be no sustainable modes of behavior that are free of pressures and stresses. But many modes of behavior are possible and some are more desirable than others. The more attractive behaviors in social systems seem possible only if we act on a good understanding of the dynamic behavior of systems and are willing to endure the self-discipline and

short-term pressures that will accompany the route to a desirable future. [8, p. 23]

These are issues that are not insurmountable and more widespread systems thinking is possible, however, the remedy may still be incubating. Systems thinking is being incorporated into the curriculum of a small but significant number of primary and secondary schools in the United States, Australia, Europe and some other places. Not necessarily as a topic in itself, but as a tool for understanding and teaching other subjects [12]. A systems view that has been absorbed at this much more fundamental level has the opportunity to innately influence the thought processes of future decision makers and has a greater chance of finding a ready ear in a systems-aware community.

It is interesting to note that where the philosophy of systems thinking has been adopted in K-12 education, system dynamics has been chosen as the practical implementation. The reason for this partnering likely lies in the rich and democratic tool set provided by system dynamics.

The tool set is rich in that various vendors offer intuitive software applications built upon system dynamic credentials that can create models at different points along the qualitative—quantitative spectrum. The user determines the level of detail. More generic, shrink-wrapped microworlds can also help people appreciate the subtle tenets of causal relationships, and show how they might be mapped into different environments [21].

Meanwhile, the tools are democratic in that the knowledge required to drive them need not rest solely in the hands of guru-like modellers. In fact, actively involving stakeholders in the system dynamics process is a critical success factor. Moreover, the system dynamics modelling package STELLA is being widely used in American primary and secondary schools, and even the more advanced iThink product contains just four fundamental building blocks.

For all this, systems dynamics can be difficult to learn, with its history in engineering and computing possibly dissuading some people.

Of course, system dynamics is not the only way of practicing systems thinking. Yet, it is the case of this paper that when compared to a representative sample of other systems methodologies, system dynamics has a number of advantages.

Methodologies such as operation research, systems analysis and systems engineering can be called systematic rather than systemic because of the methodical way they decompose a problem and then comprehensively address each component. Therefore, they are ways of dealing with detail rather than dynamic complexity, with jigsaws rather than chess

games. There is nothing intrinsically wrong in taking this approach if, for example, a meta-methodology such as TSI, points to it.

Each of the methodologies considered in this paper, except system dynamics, lack an important final step. While soft systems methodology, organisational learning, and interactive planning may produce a conceptual solution that is both desirable and feasible, in moving the solution 'into production' there still exists an unknown quantity because the solution has not really been tested. Forrester [7] has criticised this leap of faith in many methodologies.

Still, no model, not even the best system dynamics model, can perfectly predict the future. Nonetheless, simulation means our store of incomplete knowledge is at least reduced:

Simulation speeds and strengthens the learning feedbacks. Discrepancies between formal and mental models stimulate improvements in both, including changes in basic assumptions such as model boundary, time horizon, and dynamic hypotheses. [22, p. 37].

Maybe the essence of this paper is captured by John Sterman's appeal at the end of his new text book on systems thinking and system dynamics:

Be humble about what you know and listen to your critics. Strive always to make a difference. And have fun [22, p. 901].

Few other ways of thinking offer this provocation.

References

- [1] G. Burrell and G. Morgan, *Sociological Paradigms and Organisational Analysis*, Vermont: Ashgate Publishing Company, 1979.
- [2] P. B. Checkland, *Systems Thinking, Systems Practice*, Brisbane: John Wiley & Sons, 1981.
- [3] R. L. Flood and E. R. Carson, *Dealing With Complexity: An Introduction to the Theory and Application of Systems Science*, London: Plenum Press, 1993.
- [4] R. L. Flood and M. C. Jackson (eds.), *Critical Systems Thinking: Directed Readings*, Brisbane: John Wiley & Sons, 1991.
- [5] R. L. Flood and M. C. Jackson, 'Total Systems Intervention: A Practical Face to Critical Systems Thinking', pp. 321 – 337. In R. L. Flood & M. C. Jackson (eds.), *Critical Systems Thinking: Directed Readings*, Brisbane: John Wiley & Sons, 1991.
- [1] J. W. Forrester, *Principles of Systems*. Cambridge: Wright-Allen Press, 1968.
- [7] J. W. Forrester, 'System Dynamics, Systems Thinking, and Soft OR', *System Dynamics Review*, vol. 10, nos. 2 – 3, pp. 245 – 256, 1994.

- [8] J. W. Forrester, 'Counterintuitive Behavior of Social Systems' [on-line]. Available WWW: <http://sysdyn.mit.edu/sd-intro/home.html>, 1995.
- [9] A. Giddens, *Durkheim*. Glasgow: Fontana, 1978.
- [10] M. R. Goodman, 'Systems Thinking as a Language'. In D. H. Kim (ed.), *Systems Thinking Tools*, Waltham: Pegasus Communications, 1995.
- [11] A. W. Gouldner, *The Coming Crisis of Western Sociology*. New York: Basic Books, 1970.
- [12] P. L. Hopkins, 'Simulating Hamlet in the Classroom'. *System Dynamics Review*, vol. 8, Winter, pp. 91 – 98, 1992.
- [13] M. C. Jackson, 'Beyond a System of Systems Methodologies', *Journal of the Operational Research Society*, vol. 41. no. 8, pp. 657 – 668, 1990.
- [14] M. C. Jackson and P. Keys, 'Towards a System of Systems Methodologies', pp. 139 – 158. In R. L. Flood & M. C. Jackson (eds.), *Critical Systems Thinking: Directed Readings*, Brisbane: John Wiley & Sons, 1991.
- [15] T. S. Kuhn, *The Structure of Scientific Revolutions*, Chicago: University of Chicago Press, 1996.
- [16] E. Laszlo, *The Systems View of the World: A Holistic Vision for Our Time*, Cresskill: Hampton Press, 1996.
- [17] O. Mayr, *The Origins of Feedback Control*, Cambridge: MIT Press, 1970.
- [18] G. Midgley, 'The Ideal of Unity and the Practice of Pluralism in Systems Science', pp. 25 – 36. In R. L. Flood & N. R. A. Romm (eds.), *Critical Systems Thinking: Current Research and Practice*, New York: Plenum Press, 1996.
- [19] M. Parker and G. McHugh, 'Five Texts in Search of an Author: A Response to John Hassard's "Multi-Paradigms and Organizational Analysis"', *Organizational Studies*, vol. 12, no. 3, pp. 451 – 456, 1991.
- [20] G. P. Richardson, *Feedback Thought in Social Science and Systems Theory*. Waltham: Pegasus Communications, 1999.
- [21] J. D. Sterman, *People Express Management Flight Simulator* [computer software]. Banbury: Phontis Limited, 1988.
- [22] J. D. Sterman, *Business Dynamics: Systems Thinking and Modelling for a Complex World*. New York: Irwin McGraw-Hill, 2000.
- [23] L. von Bertalanffy, *General System Theory*, New York: George Braziller, 1968.

A Case for Games in Software Engineering

Craig Caulfield
Paul Maj

School of Computer and Information Science
Edith Cowan University
Email: speck@cps.net.au, s.maj@cowan.edu.au

Abstract: Computerised management simulation games have been shown to be effective learning tools in a variety of socio-economic and socio-technical environments. Originating in war-gaming and drawing on influences from a range of different fields, games have become established instructional elements in many business, military and educational institutions because of their ability to expand the notional experience of players in a safe, yet challenging, environment. However, they appear to be under represented in the field of software project management. A case is developed that a need and opportunity exists for games in this area.

Keywords: games, systems thinking, system dynamics, software engineering, anticipatory learning

INTRODUCTION

In 1979 *The Learning Report* (Botkin et al.), the culmination of two years of meetings, seminars, and discussions concerning the world problematique, was presented at a Club of Rome conference. The report saw a growing gap between a complexity of human making and a lagging development of our own capacities to deal with it, and proposed a means of bridging this gap— anticipatory learning. Anticipatory learning differs from other types of learning in that it is:

- Future-oriented. It assumes an orientation that prepares for possible contingencies and considers long-range future alternatives. “Through anticipatory learning, the future may enter our lives as a friend, not as a burglar” (Botkin et al., 1979, p. 13).
- Participative. Anticipatory learning is not possible while there is a paternalistic assumption that one group has all the answers, and will deliver these to a less-informed constituency. When issues are explored as a joint venture then solutions become “almost self-evident, are better supported, can be more readily implemented, and are less likely to generate unwanted repercussions” (Botkin et al., 1979, p. 30).

Many of the tools of anticipatory learning draw on systems thinking principles (Fulmer, 1993; Senge & Fulmer, 1993); perhaps the one with the greatest potential, games, is the subject of this paper.

Within the literature, terms such as business management game, simulation, microworld, virtual world, and the like, have been used to mean the same and different things all at once (Maier & Grossler, 2000). Within this paper, however, 'game' will be used to refer to a computer-based model of a real-world domain (business or otherwise) that supports the learning of a single user, or group of users, about some kind of socio-economic or socio-technical system.

Games of this kind draw their intellectual integrity from a number of fields including war-gaming, education theory, small group behaviour, systems analysis and operations research, and general systems theory (Raser, 1969, pp. 46 - 65). By way of illustration, the most dominant antecedent, war-gaming will be followed here.

Games have been used as learning tools in many different business, military and social environments, but appear to be under-represented in a critical modern situation—software engineering: the systematic, disciplined, and quantifiable approach to the development, operation, and maintenance of software. Despite the name, software engineering does not enjoy the same standing as the more established engineering professions. Anecdotal evidence suggests that an urgent software crisis exists (a gap between expectations of software and the product and performance actually delivered) and has been growing since the 1960s. While quantitative data proving the existence of a software crisis is thin, it might be conceded that software engineering has room to do things better. This paper presents some preliminary research that aims to make a case for the greater use of games in this field.

DEALING WITH CHANGE AND COMPLEXITY

Senge comments:

“Perhaps for the first time in history, humankind has the capacity to create far more information than anyone can absorb, to foster far greater interdependency than anyone can manage, and to accelerate change far faster than anyone’s ability to keep pace.” (Senge, 1990, p. 69).

The driving force behind this change and complexity has many sources, notably advances in communications and information technology. However, the pace and substance of change has been seen for many years. For example, Adams (1918, pp. 489 - 498) discusses an informal law of acceleration in which technology tended to double its ability roughly every ten years between 1820 and 1900. Along the same lines, Boulding has said:

“As far as many statistical series related to activities of mankind are concerned, the date that divides human history into two equal parts is well within living memory... In a very real sense the changes in the state of mankind since the date of my birth [1910] have been greater than the changes that took place in many thousands of years before this date.” (Boulding, 1964, pp. 7, 8)

Meanwhile, Toffler (1970, p. 16) has noted that it is still within living memory that agriculture, the original basis of civilisation, has lost its dominance as the primary employer of the economically active population.

If we accept, just for the length of this paper, that change is happening for whatever reasons and at whatever pace, then we should also accept that we need to deal with its consequences in some manner. There are two classic ways in which this can be done:

- Simplify reality. Look for the primitives and hierarchy of the problem domain. Seeing abstractions or commonalities and understanding how they relate to each other can help orient our thinking when confronted with something unfamiliar (von Bertalanffy, 1968; Courtois, 1985).
- Absorb the complexity or achieve a level of requisite variety (Ashby, 1956). When we are confronted with some new situation or piece of information, it is typically compared to an array of previous knowledge, our mental models. An inference process then tries to make sense of the new information by relating it to what is already known.

Simplifying reality is the first step in analysing something new; we then need to decide what to do based on what we know. Yet, in order for the later inference process to have any substance with which to work, our store of mental models, or contexts, must be sufficient:

“In order to enhance the human capacity to act in new situations and to deal with unfamiliar events, [anticipatory] learning requires the absorption of vast collections of contexts. When contexts are restricted, the probability of shock learning increases, for shock may be conceived as a sudden event that occurs outside the known contexts. Hence one task of innovative learning is to enhance the individual’s ability to find, absorb, and create new contexts— in short, to enrich the supply of contexts.” (Botkin et al., 1979, p. 24)

A way to enrich the supply of contexts is to use one of the tools of anticipatory learning: games. The argument is illustrated by the feedback diagram in the Figure 1 (Sterman, 2000, p. 34).

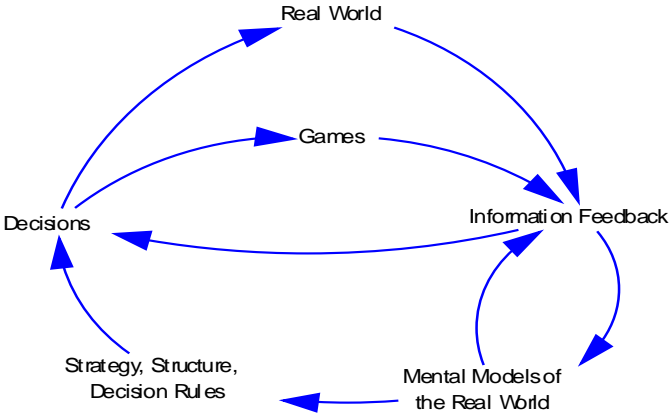


Figure 1. A learning feedback process that incorporates experimentation through games.

We receive information in its many forms from the real world in which we live. Based on this feedback, we make decisions that are filtered through our existing mental models, in the process changing or confirming the structure of our real-world systems and creating new decision rules and new strategies or reinforcing the existing. Games act as an alternative to applying our decisions to the real-world, a way of quickly and inexpensively experimenting with different policies and thereby increasing our supply of contexts. Without this tool, we must directly respond to real-world feedback that is “very slow and often rendered ineffective

by dynamic complexity, time delays, inadequate and ambiguous feedback, poor reasoning skills, defensive reactions, and the costs of experimentation” (Sterman, 2000, p. 37).

THE RATIONALE OF GAMES

An Origin in War Games

While the exact origins of war games are somewhat unclear, students of the history of chess, and similar board games played for pleasure, have noted that at an early stage such games were used as symbolic equivalents of warfare (Murray, 1913, pp. 46 - 50). For example, in the ancient Chinese game of Wei-Hai, dating from around 3000 BC, players moved coloured stones on a grid with the goal of controlling as much space as possible. While no diagrams or game pieces for Wei-Hai have survived, descriptions suggest it resembled the modern Japanese game of Go (Smith, 1998, p. 805). In the Indian game of Chaturanga (c. 1000 BC), generally assumed to be the oldest form of chess, a group of up to four players used a board divided into squares, and pieces in the shapes of elephants, soldiers, cavalry, and nobles. In contrast to Wei-Hai, the object was to capture the opponents’ pieces rather than to control territory. Chance elements were introduced by a dice.

War games continued to draw on chess and variations on the theme up to the 17th century, adding verisimilitude by using pieces shaped as soldiers, weapons of war, and royalty. Koenigspiel, or Kings Game, developed by Christopher Weikmann at Ulm, Germany in 1664 consisted of thirty pieces and a large board, but remained essentially chess. A more elaborate version of Koenigspiel, called War Chess, was developed by Dr C. L. Helwig in 1780 at the German Court of Brunswick. Helwig’s game had a playing board with 1666 squares, over 200 specialised pieces representing various military units, with the rules applied and adjudicated by an impartial game director (Wolfe, 1993, p. 449).

Around the late 18th and early 19th centuries, war-gaming experienced several developments. The games became an accepted tool of military training, particularly in Germany, and moved to a new level of complexity. For example, New Kriegspiel, developed by Georg Venturini at Schleswig in 1798 followed a 60-page rule book that defined hundreds of troop lists and supporting batteries and was played on a 3600-square surface representing the Franco-Belgian border (Wolfe, 1993, p. 449).

From its intellectual home in Germany, the war games concept spread to other countries in the latter part of the 19th century. In 1872, war games were introduced into the British Royal Artillery (Lane, 1995, p. 608). Meanwhile, the American McCarthy Little devised a war game in 1887 that used miniature battleships on maps (Macedonia, 2002, p. 36), and other war games were used extensively at West Point at the same time.

As well as being vehicles for training and education, war games were used to exercise operational plans. Germany’s Schlieffen Plan for the invasion of France in World War I was informed by war game findings (Wolfe, 1993, p. 450), and as early as 1929 Germany was gaming a various conflicts with Poland and studying the possible international reactions (Raser, 1969, pp. 47 - 48). Meanwhile, in Japan, war games conducted at the Total War Research Institute and the Naval War College allowed participants from both government and the military to experience the domestic and international factors of war.

Until the 1970s, the practice of war games has largely been physical: pieces had been moved around boards, map-based manoeuvres had added a degree of realism, and different scenarios had been played out. While computers had been used for some of the behind-the-scenes processing, they hadn't become an integral part of the war games themselves.

In 1976, then-Captain Jack Thorpe was working a research scientist in flight training at the Williams Air Force base near Phoenix, Arizona. His research was centred around improving the flight simulators used by the Air Force to initially train pilots (Sterling, 1993; Riddell, 1997). Essentially, these machines were stand-alone devices not far removed from Edwin Link's original, pre-World War II flight simulator which had itself been an amusement park ride before being adopted by the military (Macedonia, 2002, pp. 36 - 37). The simulators were also sometimes more expensive than the vehicle they emulated and ongoing running costs were exorbitant (Fullford, 1996, p. 179). Instead, Thorpe imagined a network of cheap simulators, for aircraft and other vehicles, through which military personnel could learn group skills as well as the traditional sole-operator skills (Alluisi, 1991).

At the time the technology did not exist to implement Thorpe's plan, but when he moved to the Defense Advanced Research Projects Agency (DARPA) in the early 1980s he became aware of a continuing experiment in distributed networking known as the ARPANET, the forerunner of the Internet. The means were then at hand. The eventual outcome was SIMNET (for simulator network), an interactive network of real-time, person-in-the-loop battle engagement and war-gaming (Alluisi, 1991). SIMNET was designed from the outset to be cheap and uncomplicated— factors which meant it worked and which made it highly attractive to its sponsors. From this starting point, the US military now spends some \$4b each year on simulation training and equipment (Macedonia, 2002, p. 33).

Arrival of Business Games

In 1956 the American Management Association developed what is generally considered to be the first Western business game, Top Management Decision Simulation, explicitly acknowledging its direct relation to military war-gaming:

“In the war games conducted by the Armed Forces, command officers of the Army, Navy, and Air Force have an opportunity to practice decision making creatively in a myriad of hypothetical yet true-to-life competitive situations. Moreover, they are forced to make decisions in areas outside their own specialty; a naval communications officer, for example may play the role of a task force commander. Why then, shouldn't businessmen have the same opportunity?” (Ricciardi et al. cited in Cohen & Rhenman, 1961, p. 135)

In this game teams of players managed a company that produced a single product and competed with the products of other teams. Around the same time, the RAND Corporation developed a game called Monopologs based on the supply logistics of the US Air Force (Jackson, 1959). Other similar games quickly followed. For example, Andlingers's (1958) Business Management Game sets two or three teams of players in competition within a market in which each team has a single product. The teams needed to make decisions relating to production, finance, research and development, advertising as they managed their companies from quarter to quarter.

Up until this time, business games were mostly conducted by consulting firms for the benefit of corporate decision makers and executives. However, educators were also seeing the

benefits of business games. The Top Management Decision Game developed by Schreiber, was the first business simulation game used in a university class, the business policy class at University of Washington in 1957 (Watson & Blackstone, 1989, p. 486). From this point onwards, the use of business games in industry and academia grew rapidly. By 1961 it was estimated that about 100 business games had been developed and more than 30,000 executives had played at least one game (Kibbee et al., 1961). Meanwhile, a survey of 90 American business schools found that only eight had not, and were not intending to, introduce business games into the curriculum in the near future (Dale & Klasson, 1962).

By the late 1960s and early 1970s business game penetration in business and academia had plateaued. While the tool had gained a certain degree of saturation, there were also some validity and reliability concerns (Neuhauser, 1976). But, since the 1980s there has been something of a resurgence because:

- There have been improvements in the symbols and software used to map and model system structure, for example STELLA, iThink, and Powersim.
- New ideas have been adopted from behavioural decision theory which help to transfer policymakers' knowledge into computer models. "Behavioural decision theory can help modelers to ask better questions of policymakers, to specify decision processes more accurately, and to capture more or (sic) policymakers' knowledge in maps and algebra" (Morecroft, 1988, p. 315).
- There have been improvements in methods of simulation analysis that enable modellers and model users to gain better insight into dynamic behaviour.
- Greater emphasis has been placed on small transparent models, on games and on dialogue between mental models and the tools.

Currently the state of business games is alive and growing (Burgess, 1991; McKenna, 1991; Faria, 1998).

Why Games?

To a large degree, games have been found to be more expensive and more administratively demanding than some other forms of instruction (Petranek, 1994). However, there are some significant offsetting advantages.

Boat builders, aircraft manufacturers, and engineers have long recognised that it is far cheaper to study a given phenomenon in a model in the first instance than to build the real thing (Raser, 1969, p. 15). In relation to policy formulation, where the end product may not be a physical construction, it can also be cheaper in the sense that a particular course of action can be tested in a simulation and potentially avoid costly mistakes.

A game is a visible representation of a problem space and as such has the potential to foster collective understanding. It is interesting to note that many of the tools of systems thinking, such as causal loop diagrams, rich pictures, or system archetypes, are visual rather than verbal descriptions. "A systems diagram is a powerful means of communication because it distils the essence of a problem into a format that can be easily remembered, yet is rich in implications and insights" (Goodman, 1995, p. 6). Of course, the substance of a systems diagram, or a

game, will be the mental models of those creating it. It has been noted that the human capacity to understand the implications of our mental models and to accurately trace through even a smaller number of causal relationships is fairly limited (Miller, 1956, p. 457; Simon, 1957, p. 198). But, if we are able to capture a mental model in a game, we have a tool that can be run and re-run, shared, placed under stress, and learned from.

For all these good points there are some dangers to be heeded.

For example, computers make it possible to implement games of incredible richness. Such games might be able to accommodate elements of time pressure, role-playing, systems thinking, decision-making, computer skills, random events, analysis and negotiation skills, all presented through a multi-media interface. Participants might also tend to play the game to win, as they might an arcade game, rather than to learn; alternatively, the richness of the interface may overwhelm or discourage other participants. Therefore, there needs to be an appropriate emphasis on technology and a balance of game elements at each stage (Meadows, 1989, p. 639).

For best effect, games need to be accompanied by an appropriate level of pre-game briefing and post-game debriefing (Petranek et al., 1992). Some games are deliberately vague regarding the details they supply to participants, forcing the players to work out issues as part of the experience. The post-game debrief is perhaps the most critical learning component since it is here that participants can be helped to create a comparison between the game experience and their own mental models (Simons, 1993, p. 136). “The debrief can be very important in helping people to reflect on what they have experienced, in enabling them to share and debate experiences, feelings and views, and finally, in helping them to construct their experiences into understanding which can be re-applied” (Lane, 1995, p. 616).

A CASE IN POINT: SOFTWARE ENGINEERING

In 1968 and 1969 NATO convened conferences of computer industry representatives and academics to help address what was seen as a growing gap between what was generally hoped for in complex software systems and what was actually achieved (Naur & Randell, 1969; Buxton & Randell, 1970). At the time it was recognised that the demands on software practitioners from industry, defence, and consumers would likely grow at an exponential rate. Yet, software engineering was then more of a craft than a profession (the term *software engineering* in the conference titles was considered deliberately provocative) and was already struggling to meet quality and performance measures; a software crisis in fact.

By 1982, it was estimated that 15% of all software projects failed to deliver anything, and cost over-runs of 100% to 200% were not uncommon (DeMarco, 1982, p. 3). In more recent times, the situation is still common:

“For every six new large-scale software systems that are put into operation, two others are canceled. The average software development project overshoots its schedule by half; larger projects generally do worse. And some three quarters of all large systems are “operating failures” that either do not function as intended or are not used at all.” (Gibbs, 1994, p. 86)

Despite some admirable successes (for example Copeland et al., 1995), software project and product failures tend to gather more attention (for example Sauer, 1993; Myers, 1994; Stix, 1994; Applegate et al., 1996a; Applegate et al., 1996b; Barlas, 1996; Glass, 1998, 1999) and

influence how the industry is perceived. For the most part, runaway projects such as these can be seen failures of management rather than failures of technology (DeMarco & Lister, 1987, p. 4; Schlender, 1989, p. 72; Flowers, 1996).

Glass (1994, p. 43) suggests that anecdotal software project war stories do not provide sufficient hard data to support the claim of a software crisis. Indeed:

“Next time you’re on the Internet, or making an airline reservation, or depositing money in your bank, or checking out at the grocery store, or using a credit card, or watching a space mission evolve on the evening news, or driving your fairly modern car, think about the computers and software that make all those things possible.” (Glass, 2000, p. 2)

Whichever view is individually more attractive, it may be conceded that the demands placed on the software development community will continue to grow and that room exists to improve both the quality of software and other key project management processes.

It is the argument of this paper, and the subject of continuing research of which this is just the beginning, that one way to address issues of software and project management quality is to expand the experience of software practitioners and managers through the use of games.

The software development process has been represented in computer models a number of times in the past (for example Abdel-Hamid & Madnick, 1991; Variale et al., 1994; Hansen, 1996; Lin et al., 1997; Collofello, 2000; Ruiz et al., 2001) often making use of system dynamics, but for the most part these models have been more estimating, rather than learning, tools and so do not fit the definition of a game given above. The penetration rate of these models into business and academia has also been minimal.

It was previously mentioned that a critical success factor for any game is its ability to link the experience to the user’s mental model. Adequate debriefing can help create this link, but the model or game itself can also add support:

- The game needs to be set within what Churchman (1968, p. 5) might call an environmental context. To provide this environmental context, a game should not be a stand-alone device but should ideally be a hub interlinked with other resources that enrich the game’s experience (Simons, 1993, p. 143).
- The structure of the game’s assumptions should be as simple as needed and transparent to the user.

To varying degrees, software development process models have so far failed to address these factors. Continuing research in this area is aimed at seeing how games can contribute to better software engineering practices by incorporating a richer context and more transparent structure.

CONCLUSION

Software is inherently complex, and this complexity is an essential rather than an accidental characteristic. This complexity can be attributed to several elements:

- Software needs to conform as best as possible to the arbitrary complexity imposed upon it by human institutions and systems (Brooks, 1995, p. 184). It is the usual case that these institutions and systems have been designed by different people with no underlying theme; still, software must be made to tie them together.
- Software “is pure thought-stuff, infinitely malleable” (Brooks, 1995, p. 185). This property is both seductive and dangerous: when change is needed it is likely that it will be easiest to change the software, but constant change, if not managed, can erode the integrity of the original design.
- Software is invisible and difficult to visualise. Architects or engineers have blueprints or schemas so that “contradictions become obvious, omissions can be caught. Scale drawings of mechanical parts and stick-models of molecules, although abstractions, serve the same purpose. A geometric reality is captured in a geometric abstraction. The reality of software is not inherently embedded in space” (Brooks, 1995, p. 185).

It is perhaps understandable, then, that software practitioners have struggled to keep pace with the expectations of all consumers of their product. Nevertheless, games represent an opportunity to address some of the performance issues.

References

- Abdel-Hamid, T. K. & Madnick, S. E. (1991), *Software Project Dynamics: An Integrated Approach*, Englewood Cliffs, Prentice-Hall.
- Adams, H. (1918), *The Education of Henry Adams: An Autobiography*, London, Constable & Co.
- Alluisi, E. A. (1991), The Development of Technology for Collective Training: SIMNET, A Case History, *Human Factors*, 33(3), pp. 343 - 362.
- Andlinger, G. R. (1958), Business Games - Play One!, *Harvard Business Review*, 36(2), pp. 115 - 125.
- Applegate, L. M., Montealegre, R. & Knoop, C.-I. (1996a), *BAE Automated Systems (B): Implementing the Denver International Airport Baggage-Handling System* (Case study 9-396-312), Boston, Harvard Business School.
- Applegate, L. M., Montealegre, R., Nelson, H. J. & Knoop, C.-I. (1996b), *BAE Automated Systems (A): Denver International Airport Baggage-Handling System* (Case study 9-396-311), Boston, Harvard Business School.
- Ashby, W. R. (1956), Self-Regulation and Requisite Variety, in F. E. Emery (ed.) *Systems Thinking*, Ringwood, Penguin Books, pp. 105 - 124.
- Barlas, S. (1996), Anatomy of a Runaway: What Grounded the AAS, *IEEE Software*, 13(1), pp. 104 - 106.
- Botkin, J. W., Elmandjra, M. & Malitza, M. (1979), *No Limits to Learning: Bridging the Human Gap: A Report to the Club of Rome*, Oxford, Pergamon Press.

- Boulding, K. E. (1964), *The Meaning of the Twentieth Century: The Great Transition*, New York, Harper & Row.
- Brooks, F. P. (1995), *The Mythical Man-Month: Essays on Software Engineering*, Sydney, Addison-Wesley.
- Burgess, T. F. (1991), The Use of Computerized Management and Business Simulation in the United Kingdom, *Simulation & Gaming*, 22(2), pp. 174 - 195.
- Buxton, J. N. & Randell, B., (eds.). (1970). *Software Engineering Techniques: Report on a Conference Sponsored by the NATO Science Committee, Rome, Italy, 27th to 31st October 1969*, Brussels, Scientific Affairs Division, NATO.
- Churchman, C. W. (1968), *The Systems Approach*, New York, Dell Publishing Co.
- Cohen, K. J. & Rhenman, E. (1961), The Role of Management Games in Education and Research, *Management Science*, 7, pp. 131 - 166.
- Collofello, J. (2000), University/Industry Collaboration in Developing a Simulation Based Software Project Management Training Course, in S. A. Mengel and P. J. Knoke (eds.), *Proceedings of the Thirteenth Conference on Software Engineering Education & Training*, Los Alamitos, IEEE Computer Society Press, pp. 161 - 168.
- Copeland, D. G., Mason, R. O. & McKenney, J. L. (1995), Sabre: The Development of Information-Based Competence and Execution of Information-Based Competition, *IEEE Annals of the History of Computing*, 17(3), pp. 30 - 56.
- Courtois, P. J. (1985), On Time and Space Decomposition of Complex Structures, *Communications of the ACM*, 28(6), pp. 590 - 603.
- Dale, A. G. & Klasson, C. R. (1962), *Business Gaming: A Survey of American Collegiate Schools of Business*, Austin, Bureau of Business Research, University of Texas.
- DeMarco, T. (1982), *Controlling Software Projects*, New York, Yourdon Press.
- DeMarco, T. & Lister, T. (1987), *Peopleware: Productive Projects and Teams*, New York, Dorset House Publishing Co.
- Faria, A. J. (1998), Business Simulation Games: Current Usage Levels - An Update, *Simulation & Gaming*, 29(3), pp. 295 - 308.
- Flowers, S. (1996), *Software Failure, Management Failure: Amazing Stories and Cautionary Tales*, New York, John Wiley & Sons.
- Fullford, D. A. (1996), Distributed Interactive Simulation: It's Past, Present, and Future, in J. M. Charnes, D. J. Morrice, D. T. Brunner and J. J. Swain (eds.), *Proceedings of the 1996 Winter Simulation Conference*, New York, ACM Press, pp. 179 - 185.
- Fulmer, R. M. (1993), The Tools of Anticipatory Learning, *Journal of Management Development*, 12(6), pp. 7 - 14.
- Gibbs, W. W. (1994), Software's Chronic Crisis, *Scientific American*, 271(3), pp. 86 - 95.

- Glass, R. L. (1994), The Software-Research Crisis, *IEEE Software*, 11(6), pp. 42 - 47.
- Glass, R. L. (1998), *Software Runaways*, Upper Saddle River, Prentice Hall.
- Glass, R. L. (1999), *Computing Calamities: Lessons Learned from Products, Projects, and Companies That Failed*, Upper Saddle River, Prentice Hall.
- Glass, R. L. (2000), Talk About a Software Crisis - Not!, *The Journal of Systems and Software*, 55(1), pp. 1 - 2.
- Goodman, M. R. (1995), Systems Thinking as a Language, in D. H. Kim (ed.) *Systems Thinking Tools*, Waltham, Pegasus Communications, pp. 6 - 7.
- Hansen, G. A. (1996), Simulating the Software Development Process, *IEEE Computer*, 29(1), pp. 73 - 77.
- Jackson, J. R. (1959), Learning From Experience in Business Decision Games, *California Management Review*, 1(1), pp. 23 - 29.
- Kibbee, J. M., Craft, C. J. & Nanus, B. (1961), *Management Games: A New Technique for Executive Development*, New York, Reinhold Publishing Corporation.
- Lane, D. C. (1995), On a Resurgence of Management Simulation Games, *Journal of the Operational Research Society*, 46(5), pp. 604 - 625.
- Lin, C. Y., Abdel-Hamid, T. K. & Sherif, J. S. (1997), Software-Engineering Process Simulation Model (SEPS), *The Journal of Systems and Software*, 38, pp. 263 - 277.
- Macedonia, M. (2002), Games Soldiers Play, *IEEE Spectrum*, 39(3), pp. 32 - 37.
- Maier, F. H. & Grossler, A. (2000), What Are We Talking About? - A Taxonomy of Computer Simulations to Support Learning, *System Dynamics Review*, 16(2), pp. 135 - 148.
- McKenna, R. J. (1991), Business Computerized Simulation: The Australian Experience, *Simulation & Gaming*, 22(1), pp. 36 - 62.
- Meadows, D. L. (1989), Gaming to Implement System Dynamics Models, in P. M. Milling and E. O. K. Zahn (eds.), *Computer-Based Management of Complex Systems*, Berlin, Springer-Verlag, pp. 635 - 640.
- Miller, G. A. (1956), The Magical Number Seven, Plus or Minus Two: Some Limits on Our Capacity for Processing Information, *Psychological Review*, 63(2), pp. 81 - 97.
- Morecroft, J. D. W. (1988), System Dynamics and Microworlds for Policymakers, *European Journal of Operational Research*, 35, pp. 301 - 320.
- Murray, H. J. R. (1913), *A History of Chess*, Oxford, Clarendon Press.
- Myers, M. D. (1994), A Disaster for Everyone to See: An Interpretive Analysis of a Failed IS Project, *Accounting, Management, and Information Technologies*, 4(4), pp. 185 - 201.

- Naur, P. & Randell, B., (eds.). (1969). *Software Engineering: Report on a Conference Sponsored by the NATO Science Committee, Garmisch, Germany, 7th to 11th October 1968*, Brussels, Scientific Affairs Division, NATO.
- Neuhauser, J. J. (1976), Business Games Have Failed, *Academy of Management Review*, 1(4), pp. 124 - 129.
- Petranek, C. F. (1994), A Maturation in Experiential Learning: Principles of Simulation and Gaming, *Simulation & Gaming*, 25(4), pp. 513 - 522.
- Petranek, C. F., Corey, S. & Black, R. (1992), Three Levels of Learning in Simulations: Participating, Debriefing, and Journal Writing, *Simulation & Gaming*, 23(2), pp. 174 - 185.
- Raser, J. R. (1969), *Simulation and Society: An Exploration of Scientific Gaming*, Boston, Allyn and Bacon Inc.
- Riddell, R. (1997), Doom Goes to War, *Wired*, 5(4), pp. 113 - 118, 164 - 166.
- Ruiz, M., Ramos, I. & Toro, M. (2001), A Simplified Model of Software Project Dynamics, *The Journal of Systems and Software*, 59(3), pp. 299 - 309.
- Sauer, C. (1993), *Why Information Systems Fail: A Case Study Approach*, Henley-on-Thames, Alfred Waller Limited.
- Schlender, B. R. (1989), How to Break the Software Logjam, *Fortune*, 120(7), pp. 72 - 76.
- Senge, P. M. (1990), *The Fifth Discipline: The Art & Practice of The Learning Organization*, Milsons Point, Random House.
- Senge, P. M. & Fulmer, R. M. (1993), Simulations, Systems Thinking and Anticipatory Learning, *Journal of Management Development*, 12(6), pp. 21 - 33.
- Simon, H. A. (1957), *Models of Man Social and Rational: Mathematical Essays on Rational Human Behavior in a Social Setting*, New York, John Wiley & Sons.
- Simons, K. L. (1993), New Technologies in Simulation Games, *System Dynamics Review*, 9(2), pp. 135 - 152.
- Smith, R. D. (1998), Essential Techniques for Military Modeling & Simulation, *Proceedings of the 1998 Winter Simulation Conference*, Los Alamitos, IEEE Computer Society Press, pp. 805 - 812.
- Sterling, B. (1993), War is Virtual Hell, *Wired*, 1(1).
- Sterman, J. D. (2000), *Business Dynamics: Systems Thinking and Modelling for a Complex World*, New York, Irwin McGraw-Hill.
- Stix, G. (1994), Aging Airways, *Scientific American*, 270(5), pp. 96 - 104.
- Toffler, A. (1970), *Future Shock*, London, The Bodley Head.

Variale, T., Rosetta, B., Steffen, M., Rubin, H. & Yourdon, E. (1994), Modeling the Maintenance Process, *American Programmer*, 7(3), pp. 29 - 37.

von Bertalanffy, L. (1968), *General System Theory*, New York, George Braziller.

Watson, H. J. & Blackstone, J. H. (1989), *Computer Simulation*, New York, John Wiley & Sons.

Wolfe, J. (1993), A History of Business Teaching Games in English-Speaking and Post-Socialist Countries: The Origination and Diffusion of a Management Education and Development Technology, *Simulation & Gaming*, 24(4), pp. 446 - 463.

COPYRIGHT DECLARATION

Craig Caulfield and Paul Maj 2002. We grant a non-exclusive licence to ANZSYS 2002 and its organisers to publish this document in full in the conference papers and proceedings which may include publication on the world wide web including mirror sites, CD-Rom, or in printed form. Any other usage is prohibited without the express permission of the authors. I/we assign to ANZSYS 2002, its organisers, and educational and non-profit institutions a non-exclusive licence to use this document for personal use and in courses of instruction provided that the article is used in full and this copyright statement is reproduced.

A Case for System Dynamics*

Craig W. Caulfield
S. Paul Maj

Edith Cowan University, 2 Bradford Street, Mount Lawley, Perth, WA 6050, Australia

Engineering education provides a thorough and systematic training in the design, development, maintenance and management of complex technical systems. While such education provides the necessary technical depth to graduates, many technical systems are best understood from the perspective of human and socio-economic relationships. A case in point may be Fred Brooks' law that states adding more developers to a late software engineering project will only make it even more behind schedule. Brooks' law is based on the understanding that additional, new software engineering staff will need time to come up to speed with the project and in doing so will divert the existing developers from their primary tasks. While Brooks' law is intuitively appealing, students and practicing software engineers really have no way of testing its efficacy in their particular situations. A tool to overcome this difficulty may be system dynamics. System dynamics is a systems thinking methodology for building quantitative and qualitative models of complex situations so that they can ultimately be better understood and managed. Accordingly, it can be argued, that system dynamics should be an essential part of the education of engineers from most, if not all, of the major disciplines.

INTRODUCTION

Engineering education can deliver training that is all-inclusive and systematic in the design, development, maintenance and management of intricate technical systems. Without question, such education provides the necessary technical depth to graduates. However, many technical systems are best understood from the perspective of human perceptions and also that of a wider socio-economic context. It has been well documented that the success of technical projects is quite often almost entirely dependent on these factors.

It is a curious paradox that the software industry has helped provide the means by which others have been able to automate, reengineer and economy-scale their businesses, that is, reduce the human variable, and yet remains itself very people sensitive and intensive. For example:

*A revised and expanded version of a keynote address presented at the 3rd Asia-Pacific Forum on Engineering and Technology Education, held in Changhua, Taiwan, from 8 to 11 July 2001. This paper was awarded the UICEE diamond award (first grade) by popular vote of Forum participants for the most significant contribution to the field of engineering education.

Highly skilled people with appropriate experience, talent, and training are key to producing software that satisfies user needs on time and within budget. The right people with insufficient tools, languages, and process will succeed. The wrong people (or the right people with insufficient training or experience) with appropriate tools, languages, and process will probably fail [1].

Tom DeMarco, co-author of the often-cited *Peopleware*, has found that most software development managers agree with this premise that a project's sociology will contribute more to the final outcome than the project's technology [2]. Sociology, in this context, means addressing issues such as team formation and dynamics, role assignment, hiring, motivation, workplace design, training and many other peopleware practices. However, the same managers do not conduct their projects with this regard and instead focus on that aspect they are most comfortable with: technology;

The evident reason for this is that the manager knows how to do technology, but not how to do sociology. He/she doesn't know how to manage [3].

One of the golden rules of software engineering texts maybe a case in point - Fred Brooks' informal law that states that adding more software developers to a late project will only make it later [4]. Brooks' law is based on the understanding that the new developers will need time to come up to speed with the project and in doing so will divert the existing developers from their primary and now critical tasks. While Brooks' law is intuitively appealing, students and practicing software engineers really have no way of testing its efficacy in their particular situations because such systems are difficult to model.

One possible way to address such situations is by using the systems thinking methodology, system dynamics.

System dynamics is concerned with building quantitative and qualitative models of complex problem situations and then experimenting with and studying the behaviour of these models over time. Often such models will demonstrate how unappreciated causal relationships, dynamic complexity and structural delays may lead to counter-intuitive outcomes of less-informed efforts to improve the situation. System dynamics models make room for soft factors such as motivation and perceptions so that engineering projects can ultimately be better understood and managed.

This paper presents some initial results of implementing a simple model of Brooks' law using a system dynamics modelling software package called *iThink* to support the argument that system dynamics should be an essential part of the education of engineers from most, if not all, of the major disciplines. The model is then extended beyond Brooks' exact scope to demonstrate how it might be possible to incorporate and validate soft variables alongside the more traditional variety.

SYSTEM DYNAMICS

In the late 1950s, Jay Forrester of the Sloan School of Management at the Massachusetts Institute of Technology (MIT) was asked by General Electric to review the operations of their Kentucky appliance parts plant. The company was concerned about the oscillating nature of their production cycles that often saw periods of intense activity followed by times of virtual dormancy during which workers had to be laid off. Fluctuating demand and normal business cycles did not seem to adequately explain the situation. Coming from an electrical engineering background and with a keen interest in management science, Forrester approached the problem systematically, but with just a pencil and a note pad. Starting with columns for inventory, employees and orders, and factoring in:

...the policies they were following, one could decide how many people would be hired in the following week. This gave a new condition of employment, inventories, and production [5].

Forrester's calculations amounted to a simulation of the system operating at General Electric's plant.

Stemming from this first analysis came an article for the *Harvard Business Review* in 1958 entitled *Industrial Dynamics - A Major Breakthrough for Decision Makers* with the theme being developed and expanded in the seminal work, *Industrial Dynamics* [6]. Industrial dynamics became system dynamics as it came to be used in areas other than industry.

For some time following the publication of *Industrial Dynamics*, system dynamics was used as a tool for looking at big-picture issues such as urban decay, major sociological conditions and world economics [7-9]. In more recent times, system dynamics has come back from the big end of town and has been finding a purpose for itself in a range of business and social applications. Instrumental in this change have been Peter Senge's *The Fifth Discipline* [10], and the development of intuitive, graphical software packages that have made system dynamics modelling more democratic by hiding the computer source-code look of traditional models. As a measure of this democracy, system dynamics now finds a place for itself in a number of primary and secondary schools in the United States of America, Australia and Europe, well beyond its ground zero at MIT.

To more formally define system dynamics, it could be said that it:

...is concerned with creating models or representations of real world systems of all kinds and studying their dynamics (or behaviour). In particular, it is concerned with improving (controlling) problematic system behaviour... The purpose in applying System Dynamics is to facilitate understanding of the relationship between the behaviour of the system over time and its underlying structure and strategies/policies/decision rules [11].

A key element of this definition is the need to build a computer model of the system under consideration. The model is used to help understand the patterns of change or dynamics that a system exhibits over time and to identify the conditions that cause these patterns to be stable or unstable. This knowledge of the system can then suggest what kinds of prescriptions for governing it will work and what kinds may not [12].

However, building system dynamics models demands persistence. Translating real-world information into model elements is still an inexact science - trial and error can be just as valid as considered judgement based on experience. Perhaps a useful parallel can be drawn with that other hard, inexact activity: finding object-oriented classes. Bjarne Stroustrup, the creator of C++, notes that in design and programming there are no cookbook methods that can replace intelligence, experience and good taste; *even he just tries things* [13]. The lesson for system dynamics modellers would seem to be the same: just start, try things, take advice of experienced modellers and then iterate, iterate, iterate.

Yet the effort of building a system dynamics model has some benefits including:

- Modelling brings about an understanding of the system because of the analytical and critical thinking process it calls for. It helps bring to the surface the mental models driving the current situation - those models

...that one carries around in one's head for dealing with a problem or situation. Such a model maybe based on experience or intuition, or on folklore and myth; it may be influenced by politics and a wide spectrum of human emotions [14].

Mental models may also be totally inappropriate or counter-productive, or equally priceless. But unless they are turned into something more tangible, one may never know.

- System dynamics models make room for both quantitative or hard variables, being things that can be measured directly like program size, staffing numbers or dollars spent; and qualitative or soft variables such as motivation, commitment, confidence or perceptions. Soft variables have traditionally been left out of engineering models because they are difficult to measure and their importance may have been underestimated. Yet,

...if you omit soft variables you run the risk of failing to capture something essential to driving human affairs. Leaving out something so essential is the only hypothesis that you can reject with absolute certainty! [15].

A system dynamics model can therefore be more informed about its problem space.

With a system dynamics model in hand and George Box's tongue-in-cheek caution in mind (all models are wrong, but some are useful), the model can be run. Certain variables can be held steady while others are

changed, it can be placed under stress and tested for sensitivities and leverage points. In short, the model can be experimented with to better understand the present situation and to search for alternatives for improvement. It has been stated that:

The alternatives may come from intuitive insights generated during the [initial analysis], from experience of the analyst, from proposals advanced by people in the operating system [or in the] experience, art, and skill for imagining the most creative and powerful policy alternatives [16].

Peter Senge points out that the causes of many problems

...lay in the very well-intentioned policies designed to alleviate them. These problems were actually systems that lured policy makers into interventions that focused on obvious symptoms not underlying causes, which produced short-term benefit but long term malaise, and fostered the need for still more symptomatic interventions [10].

By simulating a problem space using a system dynamics model, it is possible to potentially make more informed decisions about events beyond our bounded rationality safe from the dangers of real-world experimentation.

BROOKS' LAW

During the 1950s and early 1960s, Fred Brooks worked for IBM as a programmer and hardware architect. In 1964, he became the manager of IBM's Operating System/360 development, a large-scale and complex project intended to provide IBM's mainframe computers with a leading-edge operating system. To give an idea of the size of the project:

...the initial Windows NT project required about 1,500 staff-years of effort, but the development of IBM's OS/360, which was completed in 1966, required more than three times as much effort [17].

His experiences, frustrations and joys during this time, and his observations of the wider industry after moving to the University of North Carolina, are embodied in the collection of essays *The Mythical Man-Month* [4]. The title refers to that fundamental unit of measurement and scheduling, the man-month; a unit that Brooks believes is often misunderstood:

Cost does indeed vary as the product of the number of men and the number of months.

Progress does not. Hence the man-month as a unit for measuring the size of a job is a dangerous and deceptive myth. It implies that men and months are interchangeable [4].

His law that states adding more software developers to an already late project will only make the problem worse is based on this lack of interchangeability of manpower and time. The cause lies in two areas:

- The new developers will need to be acquainted with the overall aims of the project, its strategy and the general plan of work. During this time, the new developers will not be full contributors and will likely divert the existing developers away from their primary tasks.
- If a group of developers, n , need to coordinate their efforts with each other then the number of communication paths can be represented by $n(n-1)/2$. This represents an interaction overhead, which may be realised in the form of project meetings, technical walkthroughs and complying with any progress reporting requirements.

Brooks' law is intuitively appealing and is generally supported in the literature [14][18-20]. Writing in the 20th anniversary edition of *The Mythical Man-Month* in 1995, Brooks acknowledged that his law was outrageously simplified, yet he still felt that it was the:

...best zeroth-order approximation to the truth, a rule of thumb to warn managers against blindly making the instinctive fix to a late project [4].

Yet, turning Brooks' law into something more than a rule of thumb, it should be able to be tested whether it is a useful concept outside the large-scale big business and government projects Brooks' was most familiar with.

MODEL EXPLANATION

The following model of Brooks' law has been created using a system dynamics modelling package called *iThink*. The grammar of *iThink* consists of only four basic elements (stocks, flows, rates and connectors) and is largely intuitive so it will not be expanded upon here. Further details are provided in Appendix 1.

In addition, a range of assumptions is made that will naturally vary according to local conditions. What is important is not so much the magnitude of these assumptions in this particular instance, but that they are relevant to the problem space under consideration and that they can be changed as needed.

Looking to the model, there is a hypothetical soft-

ware development project in hand that has been estimated at 36-man months, or 6,240 hours, and must be completed within six months. To meet this deadline a staffing level of six developers has been approved. However, the project starts with only five developers, three of whom are experienced, meaning they are aware of the objectives of the project and the plan of work; and two who are new-hires. It is assumed that the new-hires will only be half as productive as their colleagues but will gradually come up to speed as they are assimilated. This transitioning from new-hires to experienced developers has been set at three months.

Recruiting is under way to bring the team up to full strength but advertising the position, assessing the applicants and making a decision all takes time. Therefore, a delay of some two months is not unreasonable [14]. At the same time, staff are likely to leave. For the purposes of this model, it is assumed that the average employment time will be nine months and, for simplicity, it is assumed that developers will not quit the team before becoming experienced developers. Figure 1 represents to model to this stage.

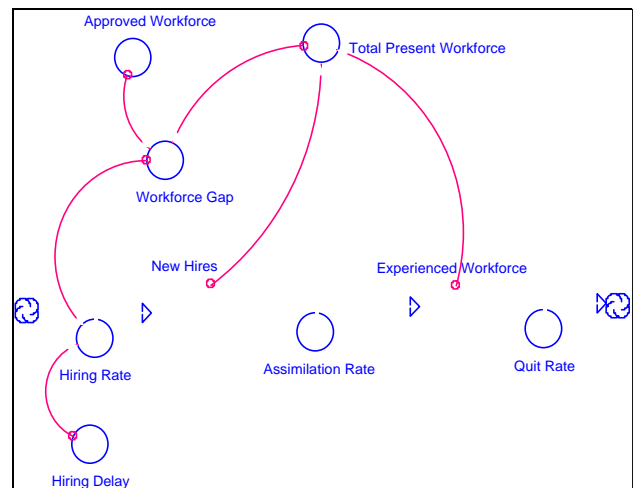


Figure 1: Model for personnel development in the project.

Staff enter the *plumbing* of the *iThink* diagram from the left, progressing to the right as they pass from being new-hires to experienced developers until they perhaps eventually leave the team. The *Total Present Workforce* will therefore be the sum of the two groups of developers. If the *Total Present Workforce* is less than the *Approved Workforce*, a *Workforce Gap* will exist and the hiring process will be initiated, subject to the prescribed delay of two months. Figure 2 represents the workflow of the project.

The team has 36 man-months of work to complete, therefore at the start of the simulation *Remaining Work* will represent this amount. Work units will flow towards *Work Completed* at a rate determined by

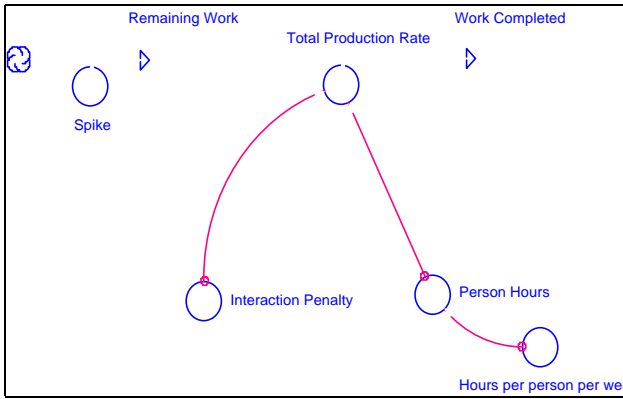


Figure 2: Workflow of the project.

the overall productivity of the team. Occasionally, there may be a spike in *Work Remaining* if the scope of the project is expanded or if the original work estimates have been found to be underestimated.

The total productivity of the team will be a function of the total workforce, the number of hours each person works per week, which has been set at a standard 40, the assumed productivity of the new-hires versus their more experienced colleagues and taking into account the interaction overhead required to coordinate all the individual development efforts. For the purposes of this model, it is assumed that the interaction overhead represents one hour per developer per week per communications path. If there are five developers, this equates to ten communications paths, and therefore ten hours per week per developer consumed in this overhead. The model in its entirety is represented by Figure 3.

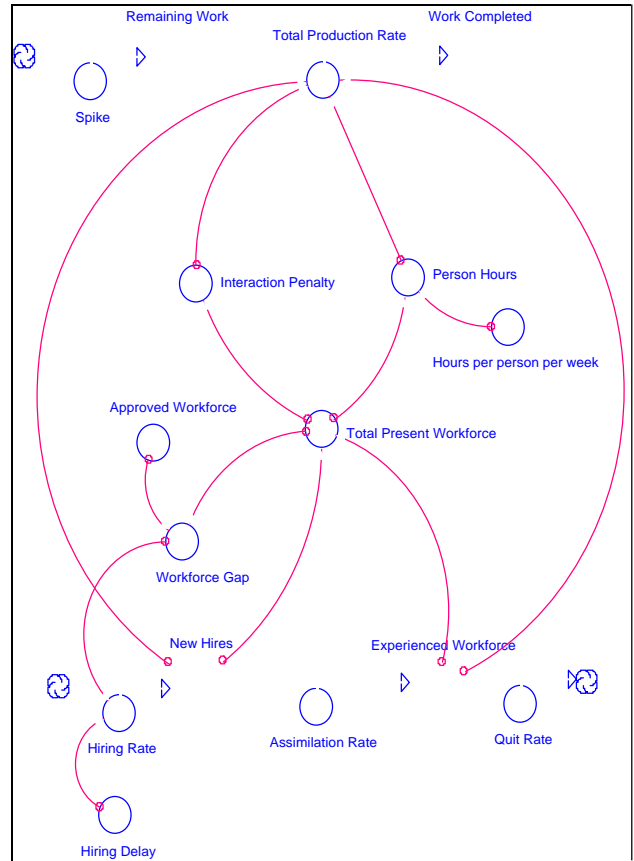


Figure 3: The model in its entirety.

MODEL RESULTS

Setting the model to run under the initial conditions described above produces the graph in Figure 4.

The approved workforce consists of six developers, but at the start of the project only five are on hand.

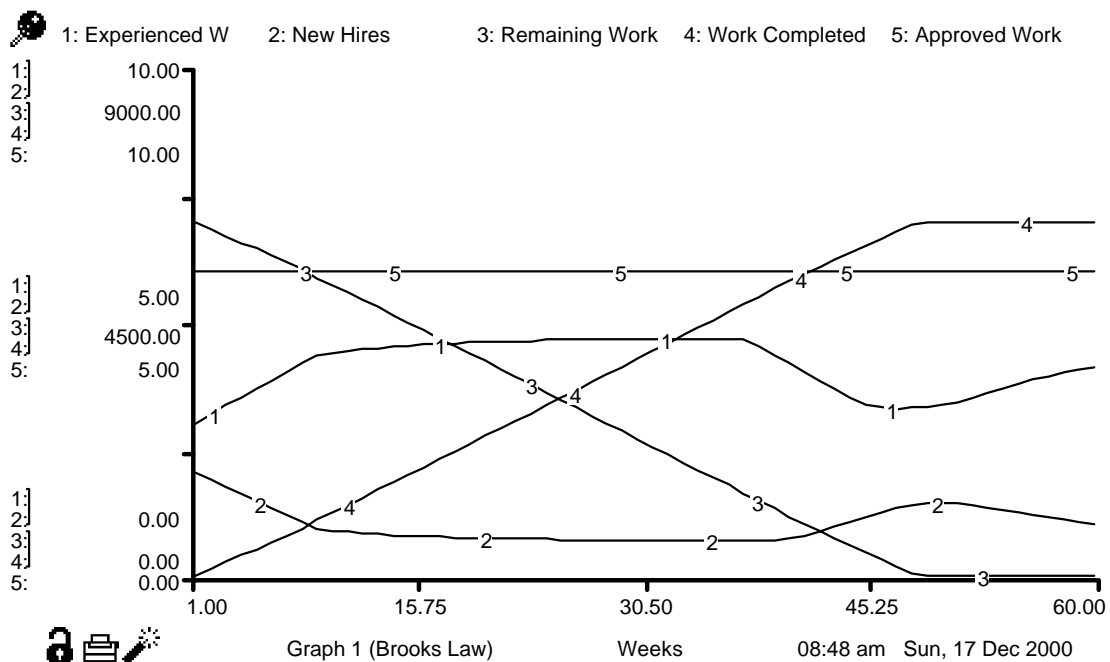


Figure 4: Graphical representation of the initial conditions.

The number of experienced workers gradually increases and the number of new hires dips as the latter come up to speed. The employment of one new developer, after the prescribed two-month delay, is masked in this transition. After nine months, or 36 weeks, experienced developers begin to leave, which initiates the hiring process again.

Even allowing for the fact that the project started with one developer less than required, the graph indicates that simply dividing the effort by the number of staff on hand will not yield an overall completion time. With the best will, the project will take nearly 12 months to complete rather than the original six.

Assume now that the project has been underway for five months, or 20 weeks, when it is discovered the original man-month estimates were understated.

Another 12 man-months of work have been assessed. Assuming 40-hour weeks and a present staff of six developers, this means the project will be extended by another eight weeks. To bring this figure down, the project manager decides to increase the approved staffing to eight developers. The resulting graph under these circumstances is shown in Figure 5.

Despite bringing on more staff, the project is still not able to hit its revised completion date and now takes nearly 18 months to complete.

ENHANCEMENTS TO BROOKS' LAW

The variables that make up the model of Brooks' law thus far are informed by the quantitative, or hard, data typical to an engineering project. Yet, it may be as

relevant to consider such a project from a socio-technical point of view, raising the need to evaluate qualitative, or soft, data. For example, soft factors may need to be considered, such as morale, commitment and knowledge levels, alongside hard factors such as headcounts, dollars spent and deliverables. This is because such factors can have an impact in areas such as productivity and hence completion times and cost.

As mentioned previously, system dynamics makes room for these soft factors. To demonstrate how this might be possible the model of Brooks' law has been extended to incorporate a number of soft variables such as occupational stress and stakeholders' perceptions of quality of the deliverables.

The relationship between occupational stress and job performance has been well documented, discussed and modelled [21-25]. A certain level of stress is

...healthy and enables employees to feel a sense of achievement and to get satisfaction from the job. However, if the amount of stress exceeds the optimum and starts to place excessive demands on the employee, the result will be lower performance. At this point, the employee loses the ability to cope, finds difficulty in making decisions and demonstrates erratic behaviour [25].

Meanwhile, the way in which clients perceive the quality of the service they receive can be considered a key performance indicator that has implications for remuneration packages and other penalties or rewards defined in service level agreements. These factors are,

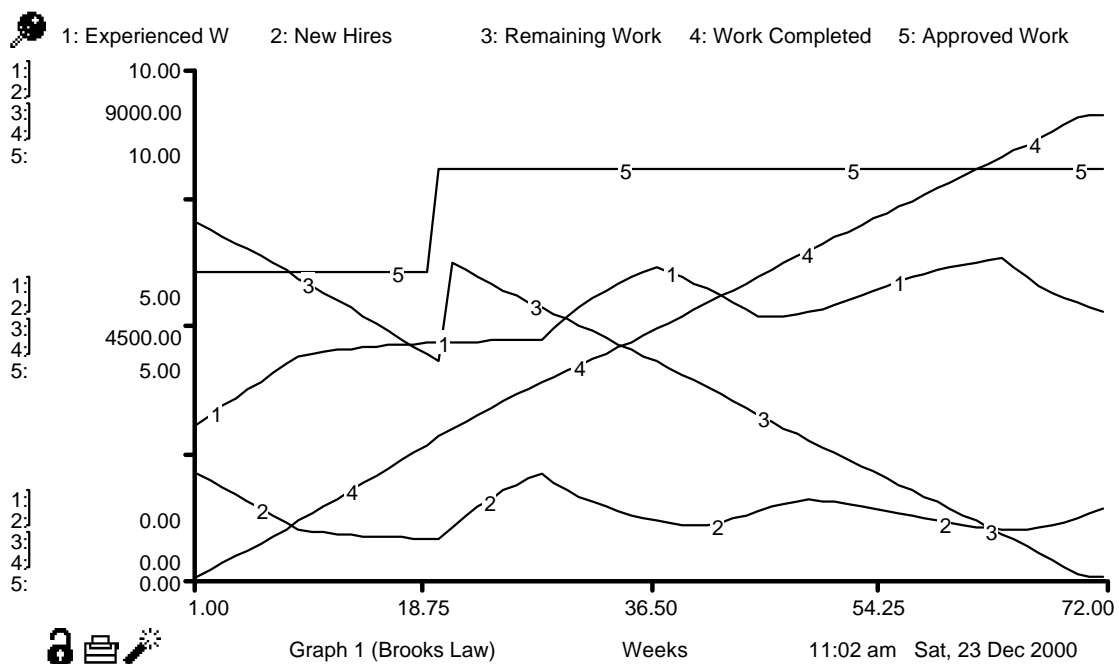


Figure 5: Graphical representation of the revised conditions.

therefore, considered as relevant additions to a model of a software development project.

The aim here is to show how it might be possible to incorporate qualitative factors, specifically perceptions of quality, into a model and then to show how it might be validated to the point where it could be used as a tool to influence policy decisions, despite lacking total quantitative comfort.

PERCEIVED AND ACTUAL QUALITY

In many human transactions there is often a gap between perception and reality. Festinger discusses something similar in *A Theory of Cognitive Dissonance* [26]. Elements of an individual's cognition (things a person knows about themselves, their behaviour, their surroundings) may deviate markedly from reality creating an uncomfortable dissonance. The cause of this dissonance may be imperfect knowledge about a situation or simply a factor of human society: *very few things are all black or all white; very few situations are clear-cut enough so that opinions or behaviours are not to some extent a mixture of contradictions* [26]. Furthermore, the dissonance may be fleeting or long-lasting; or an individual may be working to resolve and reduce the dissonance in some way or equally simply ignoring it. The result is that what we know and what we do may be inconsistent.

An example might illustrate the point. The quality of service experienced by a client, and therefore their perception of that quality, may be different for the actual quality being offered by the provider more generally. It could be the case that at a point in time, a client happened to encounter a staff member fully aware of their needs and were able to have their transaction completed quickly and efficiently. However, the rest of the provider's clients that day may not have been so lucky. Through incomplete knowledge a gap is created between perceived and actual quality.

The size of this gap can also grow, shrink and overshoot because there is often resistance, and therefore a delay, in adjusting perceptions and then taking action. A single experience of good or bad service may not cause a reaction, but an accumulation of such experiences will. The magnitude of this delay can be influenced by factors including the level of industry competition, client loyalty and mobility and the frequency of client contact [27].

Furthermore, this relationship will likely be asymmetric. When reality is less than perception, perceptions are adjusted rapidly as represented in Figure 6 (bad news travels quickly).

On the other hand, when reality is greater than perception, the adjustment time is much longer as

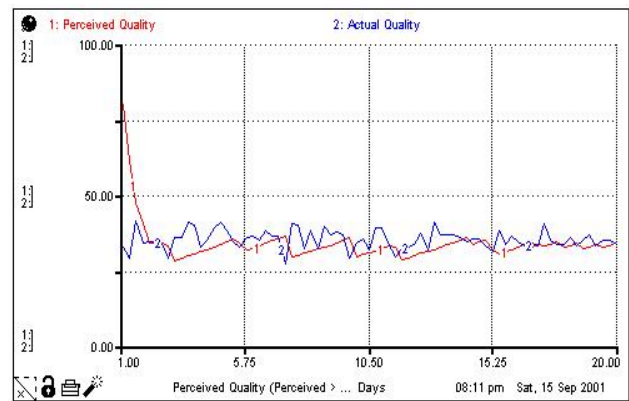


Figure 6: When actual quality is less than perceived quality, perceptions are quickly adjusted.

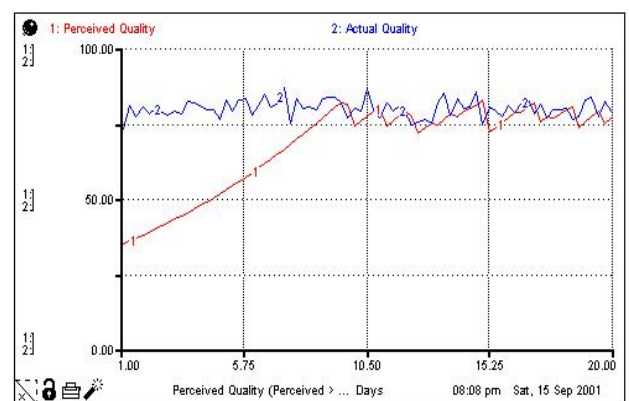


Figure 7: When actual quality is greater than perceived quality, perceptions are more slowly adjusted.

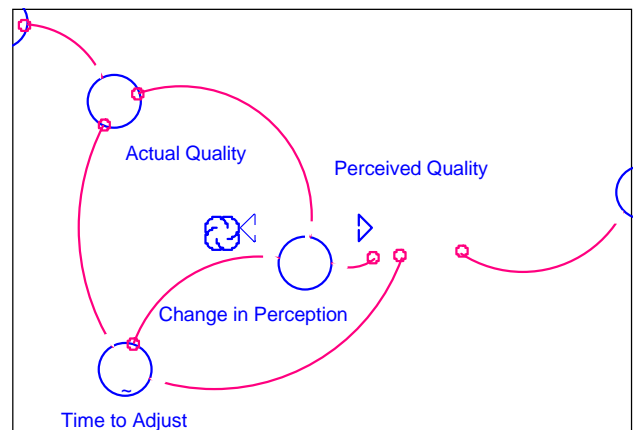


Figure 8: The service quality model component.

represented in Figure 7 (it may take ten good experiences to overcome a single bad experience).

Within the model of Brooks' law, perceived and actual quality are exhibited in Figure 8.

The factors that determine quality have many interpretations [28]. However, for the purposes of this model, actual quality is taken to be a measure of the timeliness of the deliverables and the gap between the delivered functionality and the client's requirements

[29]. The dynamics of these variables influencing actual quality are outside the scope of this fragment of the model and are only shown as a generic inflow. Meanwhile, perceived quality is taken to be a subjective gauge of how the project's clients see the service level they are receiving. The inconsistency between actual and perceived quality and their relative levels will determine the rate at which the level of perceived quality will change in line with Figures 6 and 7.

Variables such as actual and perceived quality are soft factors that cannot be measured in the same way as physical quantities. So, for the purposes of modelling, these need to be quantified instead, that is, set them against an index of some kind [15]. In this case, 0 is taken to be a total absence of quality, while 100 is taken to be total fulfilment. At the start of the model, perceived quality is set to a value between 0 and 100 representing the current circumstances. As the dynamics of the model are played out over time, the levels of actual and perceived quality may rise and fall, in turn influencing other model variables.

For example, the level of quality perceived by the project's clients may influence future remuneration contracts and have broader market implications [29][30]. Internal to the project, this perception may influence the resources devoted to testing and quality procedures [31-33]. Again, such impacts appear outside the scope of this fragment of the model are shown as a simple outflow.

VALIDATION

For those familiar with models based on more demonstrable data certainty, the treatment of soft variables such as occupational stress and perceptions of quality may seem to threaten the integrity of the final product. Yet:

As long as the purpose of your model is not to predict the numerical magnitude of particular soft variables, you can greatly benefit from including them in your models. Doing so will cause you to think in a rigorous manner about the relationships the variables bear to other variables in the system [15].

Furthermore, the particular calibration of these relationships, and therefore the behaviour of the resulting model, will depend on the individual circumstances in which it is applied. For example, the present model assumes that instances of poor service will be quickly reflected in a declining perception of the quality of that service. In an industry with few repeat clients or long delays between client contacts, the delay in adjusting perceptions may be longer.

The calibration of soft variables may also seem an arbitrary process in which the model is *made* to respond in a certain manner. However, the way in which the soft variables react must be internally consistent, that is, they must generate behaviour that matches what is observed in the actual system [15]. For example, if delivery deadlines are being consistently missed and required functionality is not being addressed, then the perceived level of service quality must decline. If the model produces behaviour contrary to this real-world pattern, then it needs to be reworked.

Sensitivity analyses designed to demonstrate internal consistency feature significantly amongst the range of tests that Forrester and Senge discuss through which a system dynamics model may be validated [34]. Importantly, these accepted tests focus more on *validating* rather than *proving* system dynamics models, on building confidence in a model's soundness and usefulness as a policy tool rather than rigorous time point predictions. The compass of a system dynamics model means that the rules by which it is validated will be slightly different.

Perhaps the ultimate test of any model is the quality of the decisions that result from it. It deserves mention that sometimes very few decisions flowed from some of the significant, early system dynamics modelling exercises [35-37]. These models tended to be large, complex and constructed by academics with only minimal involvement from the model's stakeholders beyond the initial problem definition and data collection.

Yet, as it is presently practiced, system dynamics is a very democratic and collaborative process. Sterman says that system dynamics is not a spectator sport by which he means involving the stakeholders early in the process and in doing so, giving them ownership of the model, is a critical success factor [38]. Furthermore, by making room for traditionally ignored soft variables and calibrating the variables according to real-world knowledge, by facilitating rather than creating in isolation, a more informed socio-technical model may be possible.

CONCLUSIONS

The system dynamics model of Brooks' law presented here is necessarily generic and simplified and is part of ongoing research. But, even at this level, it is one realisation of a mental model that can now be shared, discussed, calibrated according to local circumstances and (hopefully) improved upon.

The results in this case tend to support Brooks' law that adding more software developers to an

already late project will only make matters worse. However, this may not always be so. For example, using a more detailed model of Brooks' law, Abdel-Hamid and Madnick [31-33] found that if the developers are added early in the project rather than towards the end, the project will have more chance of hitting its deadlines. But, without the model, the belief that this might be so would have been without support.

Making system dynamics a part of all engineering disciplines would seem to be an incremental rather than a discontinuous step since engineers are likely to be already familiar with the benefits of building models. Typically these models have been informed by hard, quantitative data drawn from the model's domain. Also present in that domain may be softer, more qualitative, data that could be equally considered relevant to the model's outcome. System dynamics is one way of incorporating soft variables into models alongside the more traditional variety, while adding also its underlying theme that more informed socio-technical models are possible.

As a means of capturing mental models, building decision flight-simulators and communicating complex ideas at a higher level than verbal descriptions, system dynamics deserves serious consideration. In response, the methodology demands the patience to understand its concepts, nuances, and power.


REFERENCES




1. Davis, A.M., *201 Principles of Software Development*. Sydney: McGraw-Hill (1995).
2. DeMarco, T. and Lister, T., *Peopleware: Productive Projects and Teams* (2nd edn). New York: Dorset House (1999).
3. DeMarco, T., Non-technological issues in software engineering. *Proc. 13th Inter. Conf. on Software Engng.*, Austin, USA, 149-150 (1991).
4. Brooks, F.P., *The Mythical Man-Month: Essays on Software Engineering* (anniversary edn). Sydney: Addison-Wesley (1995).
5. Forrester, J.W., The beginnings of systems dynamics. Banquet talk given at the *Inter. Meeting of the System Dynamics Society*, Stuttgart, Germany, 13 July (1989), <ftp://sysdyn.mit.edu/ftp/sdep/papers/D-4165-1.pdf>
6. Forrester, J.W., *Industrial Dynamics*. Waltham: Pegasus Communications (1961).
7. Forrester, J.W., *Urban Dynamics*. Portland: Productivity Press (1969).
8. Forrester, J.W., *World Dynamics*. Portland: Productivity Press (1971).
9. Meadows, D.H., Meadows, D.L., Randers, J. and Behrens, W.W., *The Limits to Growth*. New York: Universe Books (1972).
10. Senge, P.M., *The Fifth Discipline: the Art and Practice of the Learning Organization*. Sydney: Random House (1990).
11. Wolstenholme, E.F., *System Enquiry: a System Dynamics Approach*. Brisbane: John Wiley & Sons (1990).
12. Stacey, R.D., *Strategic Management and Organisational Dynamics*. Melbourne: Pitman Publishing (1996).
13. Stroustrup B., *The C++ Programming Language* (2nd edn). Sydney: Addison-Wesley Publishing Company (1993).
14. Yourdon, E., *Rise and Resurrection of the American Programmer*. Upper Saddle River: Prentice Hall (1998).
15. Richmond, B., *Modelling "Soft" Variables*. An Introduction to Systems Thinking. Hanover: High Performance Systems, 9-1 - 9-10 (1999).
16. Forrester, J.W., System dynamics, systems thinking and soft OR. *System Dynamics Review*, 10, 2-3, 245-256 (1994).
17. McConnell, S., *After the Gold Rush*. Redmond: Microsoft Press (1999).
18. Boehm, B., *Software Engineering Economics*. Upper Saddle River: Prentice Hall (1981).
19. DeMarco, T., *The Deadline: a Novel about Project Management*. New York: Dorset House (1997).
20. Pressman, R.G., *Software Engineering: a Practitioner's Approach* (4th edn). New York: McGraw-Hill (1997).
21. Selye, H., *Stress Without Distress*. Philadelphia: Signet Books (1974).
22. Homer, J.B., Worker burnout: a dynamic model with implications for prevention and control. *System Dynamics Review*, 1, 1, 42-62 (1985).
23. Hooper, N., Coping with the modern 'madness'. *Business Review Weekly*, 17, 17, 38-42 (1995).
24. Kramar, R., McGraw, P. and Schuler, R.S., *Human Resource Management in Australia*. South Melbourne: Addison Wesley Longman (1997).
25. Stone, R.J., *Human Resource Management*. Brisbane: John Wiley & Sons (1998).
26. Festinger, L., *A Theory of Cognitive Dissonance*. Stanford: Stanford University Press (1957).
27. McIntyre, P., Loyalty not enough. *Business Review Weekly*, 22, 15 December, 104-107 (2000).
28. Crosby, P.B., *Quality is Free*. New York: Penguin Books (1980).
29. Aranda, R.R., Fiddaman, T. and Oliva, R., Quality microworlds: modeling the impact of


- quality initiatives over the software product life cycle. *American Programmer*, 6, 5, 52-61 (1993).
30. Chichakly, K.J., The bifocal vantage point: managing software projects from a systems thinking perspective. *American Programmer*, 6, 5, 18-25 (1993).
 31. Abdel-Hamid, T.K. and Madnick, S.E., *Software Project Dynamics: An Integrated Approach*. Englewood Cliffs: Prentice Hall (1991).
 32. Abdel-Hamid, T.K. and Madnick, S.E., Lessons learned from modeling the dynamics of software development. *Communications of the ACM*, 32, 12, 1426-1455 (1989).
 33. Abdel-Hamid, T.K., The dynamics of software project staffing: a system dynamics based simulation approach. *IEEE Transactions on Software Engng.*, 15, 2, 308-318 (1989).
 34. Forrester, J.W. and Senge, P.M., *Tests for Building Confidence in System Dynamics Models*. In: Legasto, A.A., Forrester, J.W. and Lyneis, J.M. (Eds), *System Dynamics*. New York: North Holland 209-228 (1980).
 35. Carlson, B.R., *An Industrialist Views Industrial Dynamics*. In: Roberts, E.B. (Ed.), *Managerial Applications of System Dynamics*. Waltham: Pegasus Communications, 139-144 (1999).
 36. Fey, W.R., *An Industrial Dynamics Case Study*. In: Roberts, E.B. (Ed.), *Managerial Applications of System Dynamics*. Waltham: Pegasus Communications, 117-138 (1999).
 37. Schlager, K.J., *How Managers Use Industrial Dynamics*. In: Roberts, E.B. (Ed.), *Managerial Applications of System Dynamics*. Waltham: Pegasus Communications, 145-153 (1999).
 38. Sterman, J.D., *Business Dynamics: Systems Thinking and Modelling for a Complex World*. New York: Irwin McGraw-Hill (2000).


APPENDIX 1: THE LANGUAGE OF *iTHINK*

Essentially, *iThink* is a language that can be used to tell a story. System dynamics models described by it use the following elements of grammar to tell their story:

Stocks, , are the nouns of *iThink*. They represent an accumulation of something at a particular point in time. The slatted stocks used in the model of Brooks' law are a special version known as conveyors. They work in the same way as regular stocks except that anything entering the conveyor rides along it for a set period of time and then leaves.

Flows,   , are the verbs of *iThink*. Stuff flows through the pipe of the flow in the direction of the arrow and at a rate determined by the flow regulator in the middle. The flow regulator is fitted with a spigot that can be conceptually tightened or loosened by other variables within the model. The cloud at the end of the flow represents the boundary of the model.

Converters, , can be thought of as adverbs that modify flows. They are often used to break out the detail of the logic, that might otherwise be buried within a flow, and might be used to represent constant values. These typically influence the behaviour of the regulators on the flows.

Connectors, , tie the other three building blocks together. They represent inputs and outputs, not inflows and outflows. Connectors do not take on numerical values: they merely transmit values taken on by other building blocks.

BIOGRAPHIES



Craig W. Caulfield graduated from Murdoch University in Perth, Australia in 1994 with a Bachelor of Science in computer science and completed a Masters of Science in software engineering in 2001 through Edith Cowan University in Perth, Australia.

He currently works as a software developer for Wesfarmers Ltd while studying towards a PhD in computer science at Edith Cowan University. His particular focus is on system dynamics and software engineering.



S. Paul Maj is a senior academic at the School of Computer and Information Science, Edith Cowan University, Perth, Australia, and also Adjunct Professor at the Department of Information Systems and Operations Management, University of North Carolina (Greensboro) in the USA.

He was previously Adjunct Professor in Computer Control Systems at the Technical University of Denmark. He is an internationally recognised authority in laboratory automation and has published a commissioned book in this field.

Sociology in Software Engineering

Craig Caulfield, Gurpreet Kohli , S. Paul Maj

Edith Cowan University, Perth, Western Australia

Introduction

The sociology of software project management is an often under-represented component in the education and professional development of software engineers even though factors such as team formation, role assignment, motivation, training, hiring, and many other peopleware¹⁸ practices have been identified many times as at least equally important to the success of software projects as the technical^{14,16,18,42,44,45,46}. The reasons for this may be two-fold: the seeming arbitrariness of the sociological factors in software development is at odds with the formal and familiar technical aspects; and the lack of suitable tools with which to model and understand human dynamics.

However, these impediments may be overcome. For example, system dynamics is a modelling approach to dynamic socio-technical problems, stemming from the work of Forrester^{20,21,22} at MIT and since developed^{36,39,43}, that allows a modeller to mix soft variables (morale, perceptions, motivations) with familiar hard variables (time, cost, resources). A system dynamics model is not so much a tool for time-point prediction, but more of an experimental device to see how certain variables might change over time under the influence of unappreciated causal relationships, dynamic complexity, and structural delays. The end result is hopefully a more informed mind set with which to manage the situation at hand¹³.

By way of illustration, this paper presents some initial results of a system dynamics model based on Frederick Brooks'¹¹ well-known informal law which warns against adding more software developers to a late project for risk of making matters worse. Brooks' law, the crystallisation of many years of practical software project experience, has been critiqued many times in the literature and generally enjoys wide support, making it a solid basis for any model of the socio-technical aspects of software project management. However, it operates at a high level of aggregation and is most often associated with large-scale software development projects. In contrast, the system dynamics model presented here creates a small-team, small-project environment more likely to be encountered by software engineers in the current market.

Brooks Law

Frederick Brooks was an IBM programmer and hardware architect who in 1964 became the manager of IBM's OS/360 development. Then and now, OS/360 was one the largest and most complex operating systems ever attempted^{6,27}, and was a significant business risk for IBM given that it would not be backward-compatible with IBM's older machines^{19,38}. Brooks' experiences on the OS/360 project and his observations of the industry in general are

collected in his book *The Mythical Man-Month*^{11,12}. The title refers to that fundamental unit of measurement and scheduling, the man-month; a unit that Brooks believes is often misunderstood:

Cost does indeed vary as the product of the number of men and the number of months. Progress does not. Hence the man-month as a unit for measuring the size of a job is a dangerous and deceptive myth. It implies that men and months are interchangeable.¹²

Because of this lack of interchangeability, Brooks' informal law states that adding more developers to a late software project in the hope of meeting a looming deadline will only make matters worse. The reason lies in the fact that software projects often cannot be broken into isolated, independent units of work, meaning that the developers need to coordinate their activities at a detailed level. Therein lies an unappreciated communications overhead. For example, if a group of n developers need to coordinate their efforts with each other then the number of communication paths can be represented by $n(n-1)/2$. Time spent navigating these paths is time not spent being directly productive.

When new developers are added to the equation, the communications overhead is amplified. The new developers are usually not immediately productive because they need to become acquainted with the overall aims of the project, its strategy and the general plan of work^{10,37}, and they possibly need to undergo some form of organisational socialisation³⁴. The best, and often only, people able to provide this training and socialisation are the existing developers, who are in the process diverted from their primary tasks.

The net result is that more time is lost in bringing the new developers up to speed and in additional coordination efforts than is gained in productive time.

Brooks' law has an intuitive appeal and has been generally supported in the literature^{7,15,17,41,45}. Writing recently, Brooks acknowledged that his law was a gross generalisation and yet, in the absence of anything more conclusive, it remained the "best zeroth-order approximation to the truth, a rule of thumb to warn managers against blindly making the instinctive fix to a late project"¹².

However, not all would agree with this assessment. For example, the effects of Brooks' law can be actively mitigated by strategies such as adding developers early in the development cycle^{3,26}, adding more developers than are expected to be needed²⁴, and ensuring that documentation, technical reviews, and a less territorial ownership of software artefacts by individual developers are used to spread the knowledge about the project^{28,42}. Raymond³⁰ even suggests that Brooks' law breaks down completely under large-scale, distributed development such as Linux.

So, what are students and practitioners to make of these different views? In many respects Brooks' law has stood the test of time but has perhaps been learned too well, becoming a mantra rather than a considered decision-making tool applicable to modern software development²⁸. This will continue to be the case until it is turned into something more concrete than a rule-of-thumb, and some of its underlying assumptions are challenged. For example, most debate around Brooks' law accepts that the communications structure of software projects is a complete graph in which all developers need to talk each other, yet this

need not be so. Creating a system dynamics model is one way of turning a rule-of-thumb into something more tangible.

System Dynamics Model Description

The model described here has been built using a system dynamics software package called iThink (High Performance Systems, <http://www.hps-inc.com/>), the components of which are described more fully in the Appendix. The model describes a hypothetical software development project and makes a range of assumptions that will naturally vary according to local conditions. What is important is not so much the magnitude of these assumptions in this particular instance, but that they can be tuned to the environment they are modelling as needed.

Figure 1 shows the Human Resources section of the model which describes the hiring, assimilation, and resignation of software developers on the project. As new developers are recruited they enter the ‘plumbing’ of the model from the left and progress from being New Hires to Midrangers, and finally to Old Hands, reflecting their growing ability as they come up to speed with the project. The average time that a New Hire will take to progress to a Midranger and then an Old Hand has been set at two and four months respectively, meaning a new developer is expected to be fully productive after a total of six months²⁴.

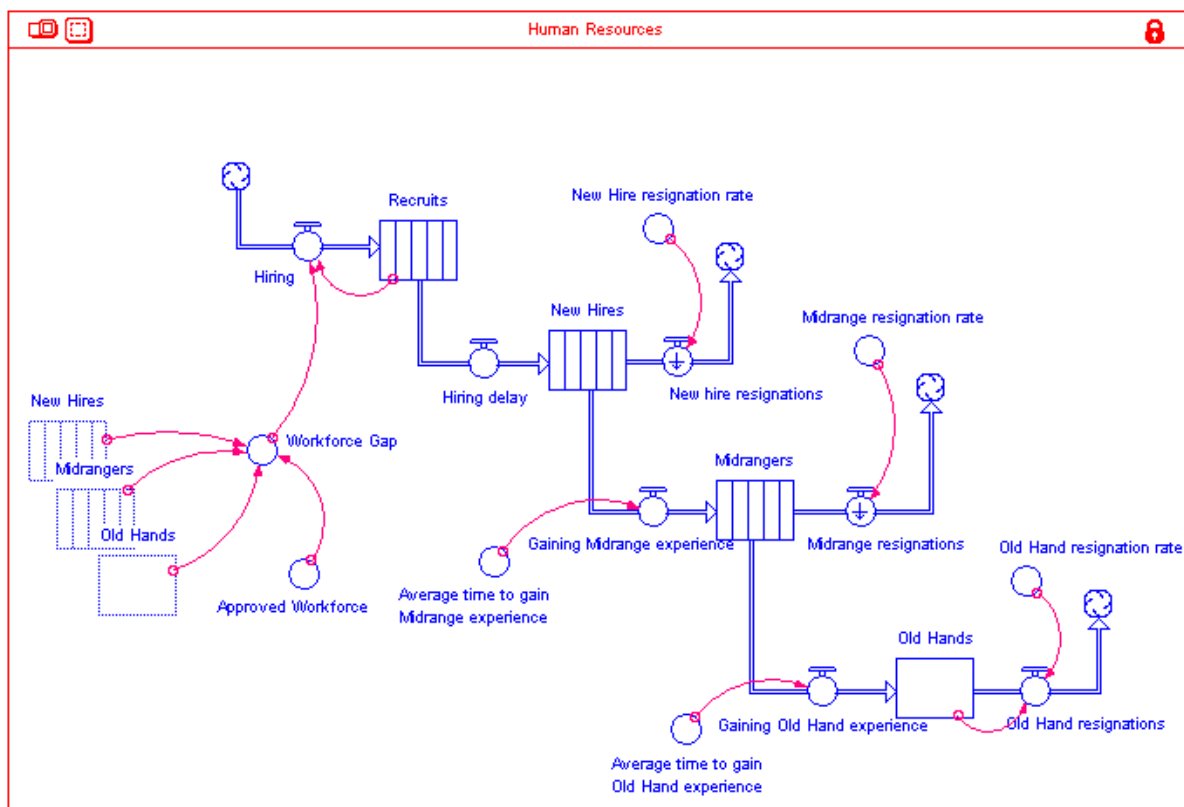


Figure 1. Human resources section of the model.

As might be expected, the project has an approved workforce level which reflects the amount of work to be done within the required time. Should the total number of developers fall below this approved level through resignations, then the process of hiring new staff is begun. However, this takes time and a delay of up to two months is not unreasonable between a position becoming available and it being eventually filled^{1,37,40}. For simplicity, it is assumed

that no New Hires will resign and the average resignation rate of Midrangers and Old Hands will be 5%^{4,5}.

Figure 2 shows the Productiveness section of the model.

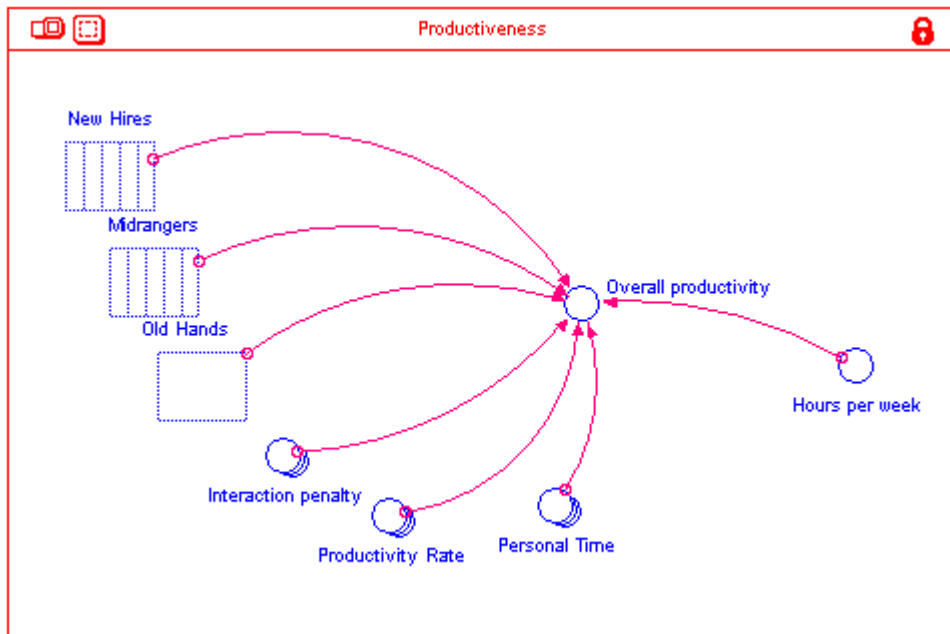


Figure 2. Productivity section of the model.

For the purposes here, productivity is considered to be potential productivity in the hours allowed during the working week, minus any losses due to faulty processes². A faulty process might be excessive administrative duties, red tape, or demands for prolonged over-time, amongst other local factors. The model here considers only three basic factors: the interaction penalty discussed by Brooks, the varying levels of productivity between the New Hires, Midrangers, and the Old Hands; and an allowance that some of each developer's day may be occupied in personal pursuits. The assumptions behind these factors are summarised in Table 1.

The project to be modelled is made up of 8 developers of varying skills levels. New Hires are considered to be working at only 50% of their capacity during the time in which it takes them to come up to speed with the project, Midrangers are working at 75% capacity, while Old Hands are considered to be as productive as possible at 95%²⁴. In addition each developer has an activity profile: net productive time during a working week is taken to be 100% of that possible, less unproductive personal time, set at a standard 10% of the working week³⁵, less the interaction penalty. The symmetric matrix to the side of table 1 represents the time in hours per week that developers spend coordinating their activities with other developers. In contrast with a key assumption behind Brooks' law, not all developers necessarily need to communicate with all other developers.

For example, Developer 1 is net productive for 77.5% of the working week, losing 10% of the week in personal time, 12.5% of the week coordinating activities with Developers 3, 4, 5, 6, and 7, and of that time is working at 50% effective capacity.

		Activity Profile (% of the working week)			Developer 1	Developer 2	Developer 3	Developer 4	Developer 5	Developer 6	Developer 7	Developer 8
	Productive Capacity	Net Productive	Personal Time	Interaction Penalty								
Developer 1	New Hire 50%	77.5	10.0	12.5	0	0	1	1	1	1	1	0
Developer 2	Midranger 75%	77.5	10.0	12.5	0	0	1	1	1	1	0	1
Developer 3	Midranger 75%	82.5	10.0	7.5	1	1	0	0	0	0	0	1
Developer 4	Midranger 75%	80.0	10.0	10.0	1	0	1	0	0	1	0	1
Developer 5	Old Hand 95%	77.5	10.0	12.5	1	1	0	0	0	1	1	1
Developer 6	Old Hand 95%	77.5	10.0	12.5	1	1	0	1	1	0	0	1
Developer 7	Midranger 75%	82.5	10.0	7.5	1	0	0	0	1	0	0	1
Developer 8	New Hire 50%	75.0	10.0	15.0	0	1	1	1	1	1	1	0

Table 1. Individual developer productive capacity and activity profiles.

The actual work to which the developers' productivity is applied is represented by the Development Work section of the model shown in Figure 3.

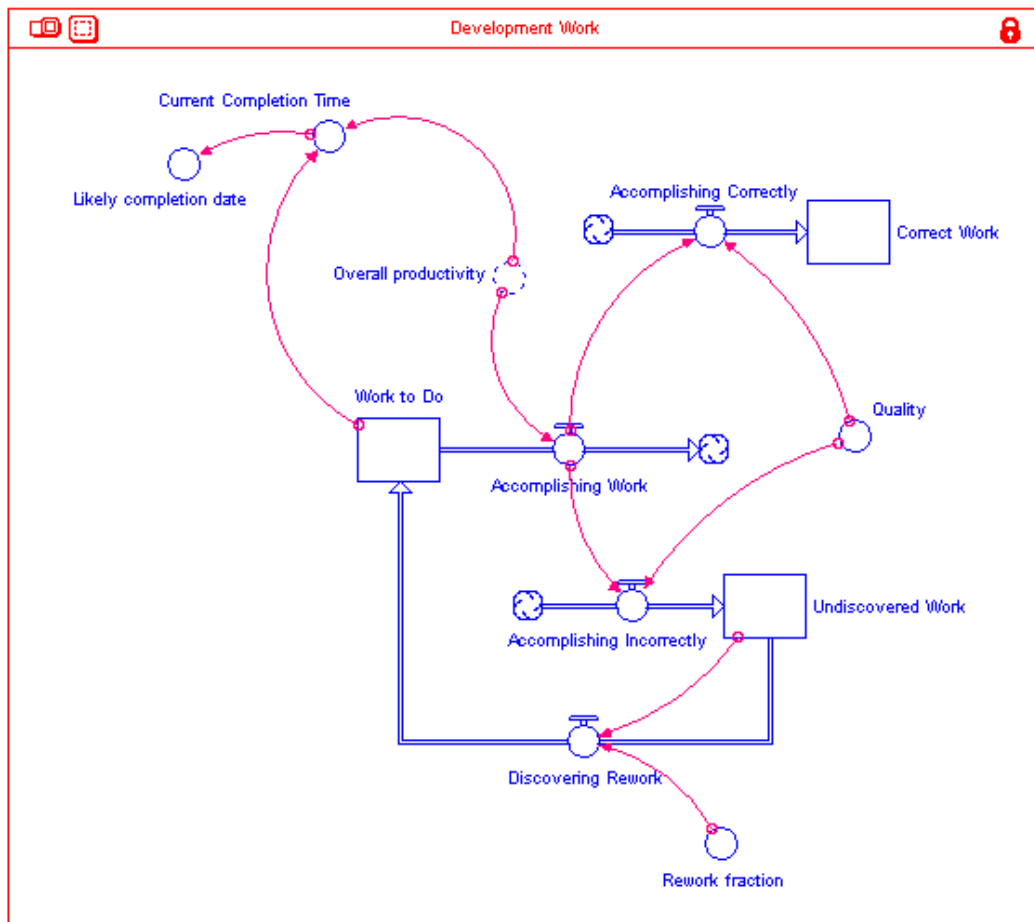


Figure 3. Development Work section of the model.

This section of the model broadly follows the classic project structure defined by Roberts^{32,33}. The project starts with a certain amount of Work to Do measured in person-months. The overall productivity of the developers is applied to reduce this work, but in the process new work may be discovered because requirements have changed or the original specifications were incomplete, and some work already done may need to be reworked because mistakes have been made. Undiscovered work and the need for rework are influenced by many factors such as schedule pressure, the presence or absence of quality control and change control mechanisms, and general management of the project. In this simple model, these extraneous factors have not been modelled, and it is considered that 10% of all completed work will need to be reworked in some way. A Likely Completion Date is calculated by dividing the Work to Do by the Overall Productivity of the developers and adding it to the time already elapsed.

The project is complete when there is no more Work to Do or Undiscovered Work.

Running the Model

The hypothetical project modelled here has been sized at 90-person months which, using accepted cost-estimation tools such as COCOMO II⁸, would take the eight developers about 12 months to complete.

Figure 4 shows the human resource numbers over a period of 24 months. The number of Old Hands gradually rises and the number of Midrangers gradually drops as the Midrangers gain experience. Likewise, the number of New Hires drops as they transition to become Midrangers.

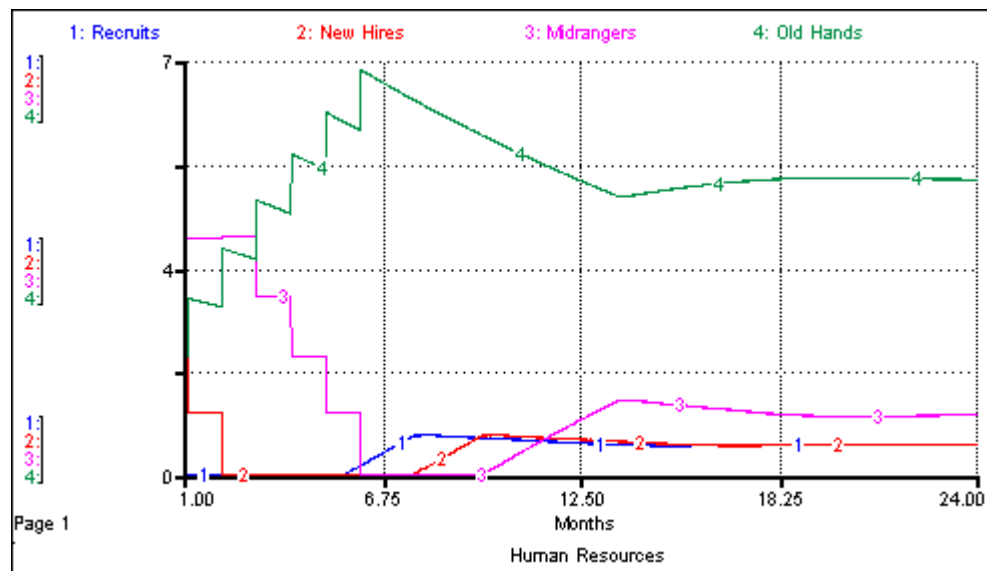


Figure 4. Human resource profile under the model's initial conditions.

Around the fourth month of the project, normal attrition (resignation of Midrangers and Oldhands) has meant that the total number of developers has fallen below the Approved Workforce of eight, and the hiring process is initiated. But, because of the hiring delay, the new developers don't make an appearance until around the seventh month.

Figure 4 also shows that the project settles down to a certain human resource profile: mainly Old Hands with a smaller number of Midrangers and New Hires, and a certain constant level of recruitment.

Under this human resource profile, the development proceeds as shown in Figure 5.



Figure 5. Development progress under the model's initial conditions.

Disturbingly, Figure 5 shows that the project will not be completed (no more Work to Do or Undiscovered Work) until just after the twenty-fourth month, double the original estimate.

To test Brooks' law, it is surmised that in the eighth month the development manager realises the project will not be completed within its scheduled period of 12 months and therefore decides to hire an additional four developers. The project under these circumstances is shown in Figures 6 and 7.

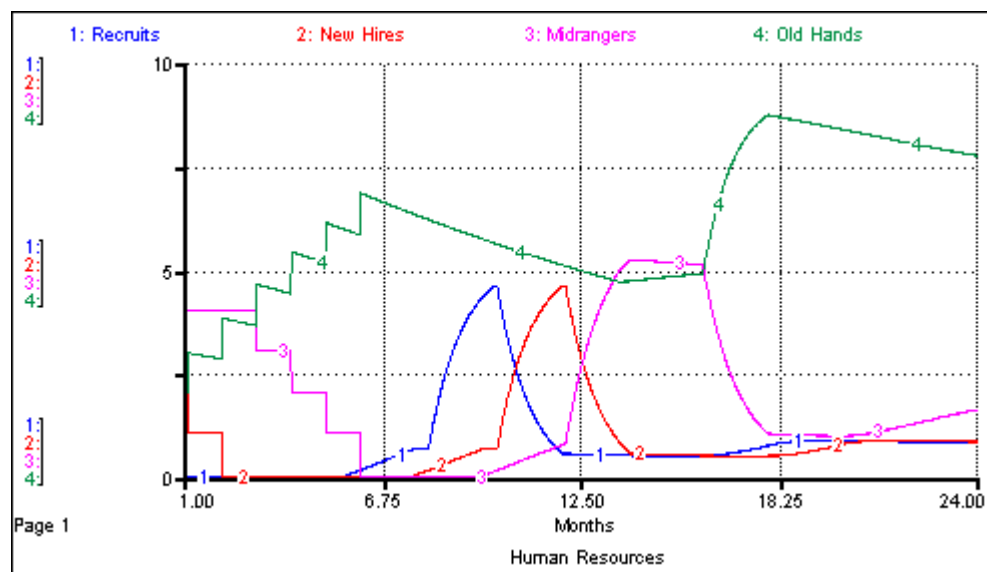


Figure 6. The human resource profile showing the hiring of four additional developers after the eighth month of the project.

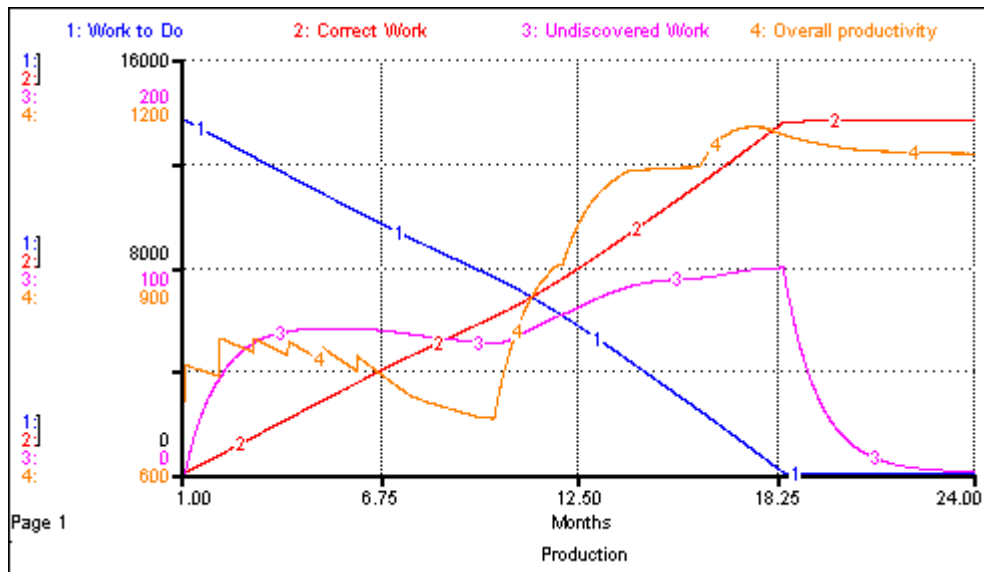


Figure 7. Development progress with four new developers joining after the eighth month.

Productivity begins to rise in the tenth month as the New Hires join the project, yet the overall effect of increasing the development workforce by 50% has been to bring the project's completion date forward only marginally. In fact, setting out to double the workforce after the eighth month has the effect of only bringing the completion date forward only one more month to that shown in Figure 7.

Given the parameters of this model, Brooks' law is not fully supported. Under a human resource profile that acknowledges that not all developers contribute equally to the project all the time, and which does not assume complete communications between all developers, perhaps Brooks' law could be rephrased as:

Adding more developers to a late project may not make the project later, but doing so will be of only marginal assistance.

Indeed, if the project had been realistically sized and resourced at the start, then the need to consider changes mid-stream may not be needed.

System Dynamics Model Validation

George Box has famously said that all models are wrong, but some are useful. The reason that models are wrong is that they are necessarily selective abstractions of reality: just as a map as detailed as the landscape it described would be as big as the landscape itself (and of no use), a model that perfectly replicated a system under study would serve no purpose⁹. Even so, models can be useful:

Models have this merit, that they do not allow us to comfort ourselves with the notion that we are following up an "idea" when we are only moving from one observation to the next in the hope that something will turn up. Too often the hypotheses with which we work are at home in the twilight regions of the mind, where their wavering outlines blend into a shadowy background. There they are safe from sudden exposure, and are free to swoop down for sustenance on whatever datum comes their way. Models are at any rate conscious, explicit, and

definite; there is nothing ghostly in their appearance or manner; they look healthy even up to the very moment of their death... The model saves us from a certain self-deception. Forced into the open, our ideas may flutter helplessly; but at least we can see what bloodless creatures they are. As inquiry proceeds, theories must be brought out into the open sooner or later; the model simply makes it sooner.²⁵

When a model becomes more than a mental model, it has a form that allows it to be shared, discussed, and hopefully improved upon; yet, it must be able to demonstrate a degree of validity for this to happen.

The compass of a system dynamics model, such as the one of Brooks' law discussed here, means that the rules by which it is validated will be slightly different from other modelling techniques. For example, the output of a system dynamics model is meant to be read, not for particular time-point predictions, but for qualitative behavioural patterns such as growth, decline, oscillation, stability, and instability²⁹. This goal of understanding general dynamic tendencies means that the model's parameters are less reliant on highly precise numerical data. Furthermore, the long-term nature of system dynamic problem statements means that parameters are likely to exceed historic ranges in any case; while the non-linear feedback structure of the models makes them less sensitive to precise parameter changes.

System dynamics models also make room for soft variables such as degrees of motivation, perception, understanding. For those familiar with models based on more demonstrable data certainty, including these soft variables may seem to threaten the integrity of the model. Yet:

As long as the purpose of your model is not to predict the numerical magnitude of particular soft variables, you can greatly benefit from including them in your models. Doing so will cause you to think in a rigorous manner about the relationships the variables bear to other variables in the system.³¹

The calibration of soft variables may also seem an arbitrary process in which the model is 'made' to respond in a certain manner. However, the way in which the soft (and hard) variables react must be internally consistent, that is, they must generate behaviour that matches what is observed in the actual system^{23,29,31}.





Conclusions

The results of this model are at variance with Brooks' law, but this might be expected because the model attempts to more realistically reflect the profile of current software development projects. For example, not all developers should be considered to be equally or immediately productive, and it need not be the case of each developer needs to coordinate their activities with each other developer. Nevertheless, the effect of adding more developers to the project only seems to help in a marginal way suggesting that there is some constraining force at work. If a software development project seems unlikely to meet its published completion date, then the common practice of adding more resources may not be the solution. Rather than attempting mid-course corrections, correctly sizing and resourcing projects from the start would appear to be a more appropriate solution and is the subject of continuing research.

While just one interpretation of the human dynamics of software project management, the system dynamics model discussed here is a means of further exploring the domain and hopefully contributing to the more rounded professional development of software engineers.

Appendix: The Language of iThink

System dynamics models described by iThink use the following grammatical elements:

- Stocks, , are the nouns of iThink. They represent an accumulation of something at a particular point in time. The slatted stocks used in the model of Brooks' law are a special version known as conveyors. They work in the same way as regular stocks except that anything entering the conveyor rides along it for a set period of time and then leaves.
- Flows, , are the verbs of iThink. Stuff (information, material, staff, money...) flows through the pipe of the flow in the direction of the arrow and at a rate determined by the flow regulator in the middle. The flow regulator is fitted with a spigot that can be conceptually tightened or loosened by other variables within the model. The cloud at the end of the flow represents the boundary of the model.
- Converters, , can be thought of as adverbs that modify flows. They are often used to break out the detail of logic, that might otherwise be buried within a flow, and might be used to represent constant values. These typically influence the behaviour of the regulators on the flows.
- Connectors, , tie the other three building blocks together. They represent inputs and outputs, not inflows and outflows. Connectors do not take on numerical values: they merely transmit values taken on by other building blocks.

Because iThink models can quickly become cluttered, any model element can be 'ghosted'. For example, in the Productiveness section of the Brooks' law model, the stocks New Hires, Midrangers, and Old Hands have been 'ghosted' (indicated by dotted outlines) rather than drawing connectors from the Human Resources section. The aim is to keep the model depiction clear and simple.

References

1. Abdel-Hamid, T. K. (1989). 'The Dynamics of Software Project Staffing: A System Dynamics Based Simulation Approach.' *IEEE Transactions on Software Engineering*, vol. 15, no. 2 (February), p. 308 – 318.
2. Abdel-Hamid, T. K. (1989). 'A Study of Staff Turnover, Acquisition, and Assimilation and Their Impact on Software Development Cost and Schedule.' *Journal of Management Information Systems*, vol. 6, no. 1 (Summer), p. 21 – 40.
3. Abdel-Hamid, T. K. and Madnick, S. E. (1991). *Software Project Dynamics: An Integrated Approach*. Englewood Cliffs: Prentice-Hall.

4. Bartol, K. M. (1983). 'Turnover Among DP Personnel: A Casual Analysis.' *Communications of the ACM*, vol. 26, no. 10 (October), p. 807 – 811.
5. Bartol, K. M. and Martin, D. C. (1982). 'Managing Information Systems Personnel: A Review of the Literature and Managerial Implications.' *MIS Quarterly*, vol. 6, no. 5 (December), p. 49 – 70.
6. Belady, L. A. and Lehman, M. M. (1976). 'A Model of Large Program Development.' *IBM Systems Journal*, vol. 15, no. 3, p. 225 – 252.
7. Boehm, B. W. (1981). *Software Engineering Economics*. Sydney: Prentice-Hall.
8. Boehm, B. W., Abts, C., Brown, A. W., Chulani, S., Clark, B. K., Horowitz, E., Madachy, R. J., Reifer, D. J. and Steece, B. (2000). *Software Cost Estimation with Cocomo II*. Upper Saddle River: Prentice Hall.
9. Bonini, C. P. (1963). *Simulation of Information and Decision Systems in the Firm*. Englewood Cliffs: Prentice-Hall.
10. Bradley, J. and McGrath, G. M. (2000). 'Boot Camp or Bordello: Whipping Rookies into Shape.' *Proceedings of the Twenty First International Conference on Information Systems*, (Brisbane), p. 467 – 472. Association for Information Systems, Atlanta, GA, USA.
11. Brooks, F. P. (1982). *The Mythical Man-Month: Essays on Software Engineering*. Reading: Addison-Wesley.
12. Brooks, F. P. (1995). *The Mythical Man-Month: Essays on Software Engineering*, 20th anniversary edition. Sydney: Addison-Wesley.
13. Caulfield, C. W. and Maj, S. P. (2002). 'A Case for System Dynamics.' *Global Journal of Engineering Education*, vol. 6, no. 1, p. 25 – 34.
14. Constantine, L. L. (1995). *Constantine on Peopleware*. Englewood Cliffs: Yourdon Press.
15. Davis, A. M. (1995). *201 Principles of Software Development*. Sydney: McGraw-Hill.
16. DeMarco, T. (1991). 'Non-Technological Issues in Software Engineering.' *Proceedings of the 13th International Conference on Software Engineering*, (Austin, Texas), p. 149 – 150. Los Alamitos: IEEE Computer Society Press.
17. DeMarco, T. (1997). *The Deadline: A Novel About Project Management*. New York: Dorset House Publishing.
18. DeMarco, T. and Lister, T. (1999). *Peopleware: Productive Projects and Teams*, 2nd edition. New York: Dorset House Publishing Co.
19. Evans, B. O. (1986). 'System/360: A Retrospective View.' *IEEE Annals of the History of Computing*, vol. 8, no. 2 (April – June), p. 155 – 179.
20. Forrester, J. W. (1961). *Industrial Dynamics*. Waltham: Pegasus Communications.
21. Forrester, J. W. (1969). *Urban Dynamics*. Portland: Productivity Press.
22. Forrester, J. W. (1971). *World Dynamics*. Portland: Productivity Press.
23. Forrester, J. W. and Senge, P. M. (1980). 'Tests for Building Confidence in System Dynamics Models.' In A. A. Legasto, J. W. Forrester and J. M. Lyneis (eds.), *System Dynamics*, (1980), pp. 209 - 228. New York: North Holland.
24. Gordon, R. L. and Lamb, J. C. (1977). 'A Close Look at Brooks' Law.' *Datamation*, vol. 23, no. 6 (June), p. 81 – 86.

25. Kaplan, A. (1973). *The Conduct of Inquiry: Methodology for Behavioral Science*. Aylesbury: Intertext Books.
26. McCarthy, J. (1995). *Dynamics of Software Development*. Redmond: Microsoft Press.
27. McConnell, S. (1999). *After the Gold Rush*. Redmond: Microsoft Press.
28. McConnell, S. (1999). 'Brook's Law Repealed.' *IEEE Software*, vol. 16, no. 6 (November/December), p. 6 – 8.
29. Meadows, D. H. and Robinson, J. M. (1985). *The Electronic Oracle: Computer Models and Social Decisions*. New York: John Wiley & Sons.
30. Raymond, E. S. (2001). *The Cathedral and the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary*. Sebastopol, California: O'Reilly & Associates.
31. Richmond, B. (1999). 'Modelling "Soft" Variables.' *An Introduction to Systems Thinking*, pp. 9-1 - 9-10. Hanover: High Performance Systems.
32. Roberts, E. B. (1962). *The Dynamics of Research and Development*. Unpublished PhD dissertation, Massachusetts Institute of Technology, Cambridge, Massachusetts.
33. Roberts, E. B. (1981). 'A Simple Model of R&D Project Dynamics.' In E. B. Roberts (ed.) *Managerial Applications of System Dynamics*, pp. 293 – 314. Waltham: Pegasus Communications.
34. Schein, E. H. (1980). *Organizational Psychology*, 3rd edition. Englewood Cliffs: Prentice-Hall.
35. Scott, R. F. and Simmons, D. B. (1975). 'Predicting Programming Group Productivity— A Communications Model.' *IEEE Transactions on Software Engineering*, vol. 1, no. 4 (December), p. 411 – 414.
36. Senge, P. M. (1990). *The Fifth Discipline: The Art & Practice of The Learning Organization*. Milsons Point: Random House.
37. Sengupta, K., Abdel-Hamid, T. K. and Bosley, M. (1999). 'Coping with Staffing Delays in Software Project Management: An Experimental Investigation.' *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, vol. 29, no. 1, p. 77 – 91.
38. Stallings, W. (1992). *Operating Systems*. New York: Macmillian Publishing Company.
39. Sterman, J. D. (2000). *Business Dynamics: Systems Thinking and Modelling for a Complex World*. New York: Irwin McGraw-Hill.
40. Stone, R. J. (1998). *Human Resource Management*, 3rd. Brisbane: John Wiley & Sons.
41. Weinberg, G. M. (1992). *Quality Software Management: Volume I Systems Thinking*. New York: Dorset House Publishing.
42. Weinberg, G. M. (1998). *The Psychology of Computer Programming*, silver anniversary edition. New York: Dorset Housing Publishing.
43. Wolstenholme, E. F. (1990). *System Enquiry: A System Dynamics Approach*. Brisbane: John Wiley & Sons.
44. Yourdon, E. (1992). *Decline and Fall of the American Programmer*. Sydney: Prentice-Hall.
45. Yourdon, E. (1997). *Death March: Managing "Mission Impossible" Projects*. Upper Saddle River: Prentice Hall.
46. Yourdon, E. (1998). *Rise and Resurrection of the American Programmer*. Sydney: Prentice-Hall.

Biographies

CRAIG CAULFIELD graduated from Murdoch University in Perth, Australia in 1994 with a Bachelor of Science in computer science and completed a Masters of Science in software engineering in 2001 through Edith Cowan University in Perth, Australia. He currently works as a senior software developer for a large Australian agribusiness while studying towards a PhD in computer science at Edith Cowan University.

GURPREET KOHLI is a PhD student at Edith Cowan University with two years of experience in Lecturing and Developing Network and Data Communication units at Edith Cowan University. Gurpreet is currently looking into web services and capacity planning of e-business sites as part of his research at Edith Cowan University.

S. PAUL MAJ is a senior academic at the School of Computer and Information Science, Edith Cowan University, Perth, Australia, and also Adjunct Professor at the Department of Information Systems and Operations Management, University of North Carolina (Greensboro) in the USA. He is an internationally recognised authority in laboratory automation and has published a commissioned book in this field.

Come Play

Craig Caulfield
S. P. Maj

School of Computer and Information Science
Edith Cowan University
Perth, Western Australia

Abstract- Games have been used as learning tools in many different business, military and social environments, but appear to be under-represented in a critical modern situation—software engineering: the systematic, disciplined, and quantifiable approach to the development, operation, and maintenance of software. Despite the name, software engineering may not enjoy the same standing as the more established engineering professions. Anecdotal evidence suggests that an urgent software crisis exists (a gap between expectations of software and the product and performance actually delivered) and has been growing since the 1960s. While quantitative data proving the existence of a software crisis is thin, it might be conceded that software engineering has room for improvement. This paper presents a case that the use of games as a research tool in software engineering needs to be more fully explored and an opportunity exists to use games to tackle some of the current issues in the field.

I. INTRODUCTION

In the science fiction novel *Ender's Game* [1], alien attacks on the Earth prompt the world government to begin training children as future military commanders by setting them to play continuous tactical and strategic computer war games. Amongst the current crop, one player stands out, Ender Wiggin—he wins every time by learning and adapting to the ever-more ridiculous challenges his teachers confront him with. For his success, Ender is selected to lead a small team of children in a complex computer battle simulation of a final confrontation with the aliens; a confrontation that ends when the aliens' home planet is destroyed. Only later is it revealed that Ender and his fellow players have been commanding real weapons, and the alien treat is suddenly no more.

As Ender and his friends realise, games can be more than play and have the potential to show us things we don't already know. These two facets of games raise the possibility that games can be considered as valid research tools.

Some definitions are first in order as the word 'game' is often used carelessly and interchangeably with model and simulation [2]. To start, a model can be described as "a miniature representation of a complex reality. A model reflects certain selected characteristics of the system it stands for. A model is useful to the extent that it portrays accurately those characteristics that happen to be of interest at the moment" [3].

A model is almost guaranteed to be incomplete because it is an abstraction of reality; the exact components of reality included in the model will depend on what the user is trying

to explore. Even so, a model "saves us from a certain self-deception. Forced into the open, our ideas may flutter helplessly; but at least we can see what bloodless creatures they are. As inquiry proceeds, theories must be brought out into the open sooner or later; the model simply makes it sooner" [4].

Meanwhile, a simulation is a special kind of model that exhibits processes in some way like the system it represents, and that shows how these processes change from state A to state B, between two points in time [5].

Finally, to play a game, "is to engage in activity directed towards bringing about a specific state of affairs, using only means permitted by specific rules, where the means permitted by the rules are more limited in scope than they would be in the absence of the rules and where the sole reason for accepting such limitation is to make possible such activity" [6].

Therefore, as used here, a model is a convenient representation (in words, numbers, or other symbols) of some real-world, socio-economic or socio-technical system; a simulation is dynamic, operational model through which changes over time are revealed; and a game is a simulation that is purposefully run, wholly or partly determined by players' decisions, within some predetermined circumstances.

Serious games, that is, games beyond simple amusement, draw their intellectual integrity from a number of sources including educational theory [7-10], operations research [11, 12], small-group behaviour research [13, 14], war-gaming, decision sciences, and systems engineering [15]. The aim of this paper is to show that games have a maturity, breadth, substance, and a role in research beyond the unprepossessing name.

II. THE VALUE OF GAMES AS RESEARCH TOOLS

To a common extent, games have been found to be more expensive and administratively demanding to develop and use than some other forms of instruction or research [16-21]. Still, there are some offsetting advantages.

For example, it has been noted that the human capacity to understand the implications of our mental models and to accurately trace through even a smaller number of causal relationships is fairly limited [22, 23]. Yet, a game is a visible and physical representation of a problem space; a captured mental model. As such, they are places to trial new ideas and to experiment with established theories [24, 25]; to replay these theories as many times as needed; places where time and space can be contracted or expanded [15];

places where it is acceptable just to try different things and where more might be learned from failure than success [26].

Even so, there are some dangers to be heeded when using games. Games are just... games, and as such are just one representation of how the world works. Therefore, "it is potentially dangerous to have players leave the gaming environment with the belief that the strategies that were effectively employed in playing the game are directly transferable to the real world" [19]. Participants should ideally be provided with more information than just the game to help them wisely discriminate between what may or may not work outside the game itself [27].

Further, the favoured contemporary medium for games, computers, make it possible to implement games of incredible richness. Such games might be able to accommodate elements of time pressure, decision-making, analysis and negotiation skills, all presented through a multi-media interface. In this situation, participants might also tend to play the game to win, as they might an arcade game, rather than to learn [18, 28]. Alternatively, the richness or over-design of the game may overwhelm or discourage other participants [17, 27]. Therefore, there needs to be an appropriate emphasis on technology and a balance of game elements at each stage [29].

The balance of a ledger of pros and cons of games given so far may still not be enough to convince some of the value of games. However, such sceptics may be demanding of games a certitude that even more mature tools and fields cannot demonstrate:

"Laboratory tests, in situ tests, and the analysis of natural analogs are all forms of model confirmation. But no matter how many confirming observations we have, any conclusion drawn from them is still an example of the fallacy of confirming the consequent. Therefore, no general empirical proposition about the natural world can ever be certain. No matter how much data we have, there will always be the possibility that more than one theory can explain the available observations" [30].

Moreover, how we understand a tool's relationship to the real world can have a bearing on how we perceive its utility:

"A model, like a novel, may resonate with nature, but it is not a "real" thing. Like a novel, a model may be convincing— it may "ring true" if it is consistent with our experience of the natural world. But just as we may wonder how much the characters in a novel are drawn from real life and how much is artifice, we might ask the same of a model: How much is based on observation and measurement of accessible phenomena how much is convenience? Fundamentally, the reason for modelling is a lack of full access, either in time or space, to the phenomena of interest" [30].

In contrast, others [15, 31] believe that this traditional test of model, simulation, or game validity— isomorphism to the reference system— is both inappropriate and inadequate. A more fruitful way to judge validity is in terms of its usefulness in reaching particular goals:

"'Usefulness' does not provide an objective standard for measurement, but allows each scholar to answer the validity question for himself within the context of his own experiences and purposes. While such an individual standard raises the spectre of non-scientific, subjective evaluation, it should be borne in mind that the 'objective truth' implied in asking for isomorphism is also a phantom. Our knowledge of the 'objective' world is too limited, our measures and our senses too fallible, our theories too uncertain and contradictory, and our data too ambiguous to justify and claims to real objectivity, no matter what means we might use to determine validity" [15].

Therefore, the value of a game depends on its stated objectives, which in turn depends on its purpose [19, 32].

III. AN EXAMPLE: THE BEER GAME

To demonstrate the utility of games as a research tool, a session of the Beer Game was held amongst teams of post graduate students from Edith Cowan University. The Beer Game is a simulation of a four-point distribution chain, originally developed at MIT and now used widely as a management educational tool in a variety of academic and commercial settings [33-36].

Each team of players takes one of the roles in the distribution chain— either retailer, wholesaler, distributor, or factory. The retailer interacts with a fictitious customer who buys cases of beer. The retailer draws cards from a deck at the beginning of each time period to find the customer demand and then orders stock from the wholesaler accordingly to maintain their inventory. To simplify things, there are no lost sales: if there is insufficient stock on hand, the retailer creates a backlog and fills the order when replenishment stock arrives.

The wholesaler receives downstream orders from the retailer and places upstream orders with the distributor. Again there are no lost sales and a backlog is created if necessary and filled when stocks become available.

The distributor interacts with the wholesaler and the factory in a similar way.

The factory is only concerned with production scheduling and since raw materials are always made available, the factory will produce as much beer as asked for.

To simulate the realities of the market place, there are shipping and order processing delays between the four points of the distribution chain. This means that orders will not be filled immediately and each point in the distribution chain will need to factor this delay into the ordering decisions.

The central theme of the game is inventory management— the price of the beer isn't important. Since the inventory carrying cost is set at \$0.50 per case, per week, and the out-of-stock cost is \$1.00 per case, per week, the aim is to maintain the minimum level of inventory to meet demand, taking care not to hold too much stock, while also ensuring that a backlog doesn't build up. To simulate market conditions, none of the teams were allowed to confer with other teams. In the real world, the retailer for example,

might handle dozens of products, supplied by many distributors, and the distributor in turn could supply many retailers. So, for the most part, upstream communication happens only through orders. Therefore, the teams were operating with perfect local knowledge but imperfect global knowledge.

The winner of the game is the team with the lowest overall inventory cost. The game was started in a state of equilibrium with four cases of beer in each of the shipping delays and customer orders at the retailer running at four cases per week. At the fifth week, customer orders at the retailer jumped to eight cases and remained at this level for the remainder of the game. The impact of this buying decision is reflected in the net inventory and ordering graphs of the teams shown in Figs. 1 to 4.

The graphs show that in response to the jump in retail sales in the fifth week, increased upstream orders were placed. Because of the shipping delays, these orders could not be filled immediately, and expensive backlogs resulted. To compensate for the backlogs and to provide a buffer, each supply chain point began placing larger upstream orders. When eventually, the backlogged orders began to be filled, so much beer was already in the supply chain that all points along the supply chain became grossly overstocked and had to drastically adjust their orders.

Interestingly, this boom-and-bust cycle was brought about by a single step in retail demand for beer in the fifth week of the game, after which this demand remained constant for the rest of the game.

The results in this instance of the Beer Game follow the pattern shown in many previous game sessions at different institutions and organisations [36-38]. Also in common with these other games sessions, the participants reported a sense of having little control over their ordering decisions and tended to see the root cause of their inventory problems as being caused by other points in the supply chain.

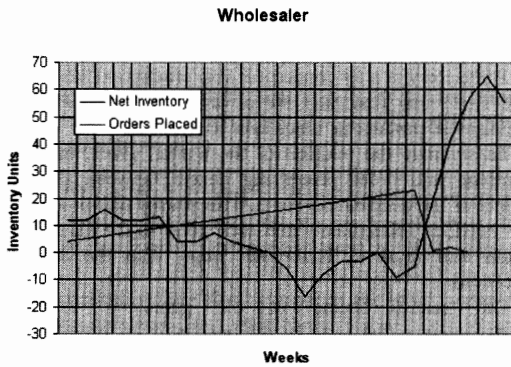


Fig. 1. Weekly net inventory and orders placed by the wholesaler.

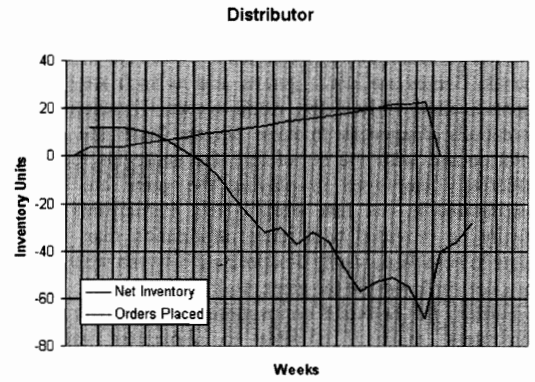


Fig. 2. Weekly net inventory and orders placed by the distributor.

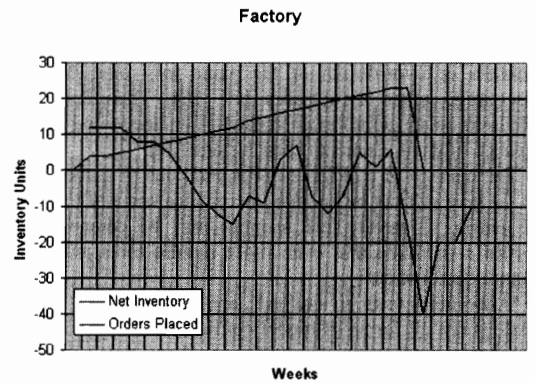


Fig. 3. Weekly net inventory and orders placed by the factory.

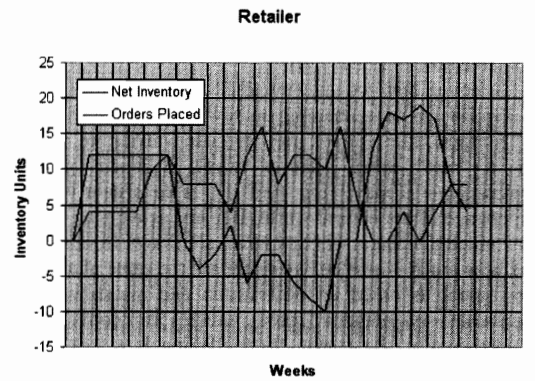


Fig. 4. Weekly net inventory and orders placed by the retailer.

IV. FURTHER RESEARCH

The Beer Game demonstration is part of an ongoing research project into ways that games can be used to understand a critical modern situation— software

engineering: the systematic, disciplined, and quantifiable approach to the development, operation, and maintenance of software. Despite the name, software engineering may not enjoy the same standing as the more established engineering professions. Anecdotal evidence suggests that an urgent software crisis exists (a gap between expectations of software and the product and performance actually delivered) and has been growing since the 1960s. While quantitative data proving the existence of a software crisis is thin, it might be conceded that software engineering has room to do things better.

Bearing in mind the definitions of model, simulation and game given in the introduction, it can be said that the software development process has been modelled [39-42] and simulated [43-49] many times. However, games have been used much less frequently with perhaps just a few examples [50, 51]. The reason for this game imbalance within the field, and between software development and other fields needs to be more fully explored.

The existing models and simulations in the field have recognised the utility of applying disciplined, systematic, and quantifiable methods to software development, that is, they describe situations of software projects managed according to software engineering principles. As such, these models and simulations would naturally serve as a starting point for game development, yet many have not presented the field's body of knowledge in an intuitive way. For example, perhaps the most well-known simulation [43] contains over 300 underlying variables, doesn't have a way to interact with the model except through direct manipulation of these variables, and yet does not describe the development process in detail [52].

Set against these models and simulations is research that has compared the learning outcomes between a range of simple and more complex games. While the most complex game offered "the richest learning experience available, the game's very formidable appearance probably intimidated a number of players or forced them into a learning situation they were unprepared or unwilling to negotiate" [53]. The next most effective game in Wolfe's study was found to be the least complex, supporting similar research that showed relatively simple games can provide essentially the same benefits as the more complex [54-56]. Therefore, making games only as complex as absolutely necessary, or hiding unnecessary detail, could be a way of achieving the best learning outcomes while avoiding the player mortality (boredom and dropout) noted by Wolfe.

Furthermore, software is often developed within an ill-structured environmental context [57-63] that includes sometimes contradictory human and business priorities, meaning that a purely technical or logical model or simulation is not always the best guarantee of success. (One way to provide this context is to have the game function as a hub interlinked with other resources and issues, rather than a stand-alone device [64-66]).

To address some of these issues, a game named Simsoft is in development by the authors. Within Simsoft, teams of

players will given a simulated software project to operate from start-up to the delivery of its objectives. Based on the starting scenario of the game, information provided during the game, and their own real-world experience, the players will make decisions about how to proceed— whether to hire more staff or reduce the number, whether to purchase certain equipment, what hours should be worked, how much time will be devoted to training, and so on. After each decision set has been entered by the players, the game will be run for its next time slice, (a day, week, month, quarter— whatever the game designer has determined). The game will then be in a new state which the players must interpret from the reports or dials or other interfaces the game provides. A fresh set of decisions can then be entered and the life of the simulated project continues.

Specifically, Simsoft will represent software project experiments much like those imagined by DeMarco [67].

Naturally, software-engineering-based games are not the only way in which a contribution could be made to better software project management. In recent years a number of organisations and academic institutions have initiated training programs that aim to more fully round the theoretical background of software practitioners, often mixing this with real-world projects [68, 69]. Yet, for the most part "students are learning their real world awareness in industry, working on real projects where their mistakes affect all around them" [70]. Games represent one way of combining theory and practice while mitigating the inherent dangers.

V. CONCLUSIONS

In many fields, games are a mature tool for instruction and research. For example war games have a lineage stretching back many thousand of years, or, if we apply more strict criteria, at least as far back as the seventeenth century [71]. Meanwhile business games, such as the Beer Game, have been informing, frustrating, and delighting managers, students, and researchers since the 1950s. In addition, information technology now makes it possible to develop games of incredible richness and verisimilitude. Despite this maturity in other fields, games have been less used in exploring issues around software development, presenting an opportunity to find out why this might be so and whether a suitable game can address current concerns.

However, applicable to any field in which games might be used, particularly those in which games are delivered via computers, Simon [72] asks a pertinent question: can a computerised representation of the real world, such as a simulation or game, ever tell us anything that we do not already know? Taking for granted two common and plausible assertions about such computerised representations— they can be no better than the assumptions built into it, and a computer can only do what it is programmed to do— then the answer to Simon's question might be, no. Yet, Simon argues otherwise. For example, even if we start with correct premises, it may be hard to infer meaning because humans generally have trouble

tracing through even a small set of causal relationships. A disinterested computerised representation is better able to do this. It may also be the case that in some circumstances, for some players, games may *only* tell us what we do not already know:

"It appears that for children, games are more than a caricature of life; they are an introduction to life—an introduction to the idea of rules, which are imposed on all alike, an introduction to the idea of playing under different sets of rules—that is, the idea of different roles, an introduction to the idea of aiding another person and of knowing that one can expect aid from another, an introduction to the idea of working toward a collective goal and investing oneself in a collectivity larger than oneself" [73].

Therefore, games may represent a serendipitous addition to the tool bag of researchers.

REFERENCES

- [1] O. S. Card, *Ender's Game*. New York: Tom Doherty Associates, 1985.
- [2] F. H. Maier and A. Grossler, "What Are We Talking About? — A Taxonomy of Computer Simulations to Support Learning," *System Dynamics Review*, vol. 16, pp. 135 – 148, 2000.
- [3] T. DeMarco, *Controlling Software Projects*. New York: Yourdon Press, 1982.
- [4] A. Kaplan, *The Conduct of Inquiry: Methodology for Behavioral Science*. Aylesbury: Intertext Books, 1973.
- [5] J. G. Miller, *Living Systems*. New York: McGraw-Hill Book Company, 1978.
- [6] B. Suits, "What is a Game?," *Philosophy of Science*, vol. 34, pp. 148 – 156, 1967.
- [7] J. Dewey, *Experience and Education*. New York: Collier Books, 1938/1963.
- [8] S. Papert, *Mindstorms*. Brighton, Sussex: The Harvester Press, 1980.
- [9] D. A. Kolb, *Experiential Learning: Experience as the Source of Learning and Development*. Englewood Cliffs: Prentice-Hall, 1984.
- [10] K. Lewin, *Field Theory in Social Sciences*. London: Tavistock Publications Ltd, 1952.
- [11] C. J. Thomas and W. L. Deemer, "The Role of Operational Gaming in Operations Research," *Operations Research*, vol. 5, pp. 1 – 27, 1957.
- [12] A. Wilson, *The Bomb and the Computer*. London: Barrie & Rockliff, The Cresset Press, 1968.
- [13] J. L. Kennedy, "The System Approach: A Preliminary Exploratory Study of the Relation Between Team Composition and Financial Performance in Business Games," *Journal of Applied Psychology*, vol. 55, pp. 46 – 49, 1971.
- [14] J. L. Kennedy, "Simulation Study of Competition in an "Open World"," *Journal of Applied Psychology*, vol. 55, pp. 42 – 45, 1971.
- [15] J. R. Raser, *Simulation and Society: An Exploration of Scientific Gaming*. Boston: Allyn and Bacon Inc, 1969.
- [16] C. F. Petranek, "A Maturation in Experiential Learning: Principles of Simulation and Gaming," *Simulation & Gaming*, vol. 25, pp. 513 – 522, 1994.
- [17] C. C. Abt, *Serious Games*. New York: The Viking Press, 1970.
- [18] J. M. Kibbee, C. J. Craft, and B. Nanus, *Management Games: A New Technique for Executive Development*. New York: Reinhold Publishing Corporation, 1961.
- [19] H. J. Watson and J. H. Blackstone, *Computer Simulation*, 2nd edition ed. New York: John Wiley & Sons, 1989.
- [20] K. J. Cohen and E. Rhenman, "The Role of Management Games in Education and Research," *Management Science*, vol. 7, pp. 131 – 166, 1961.
- [21] E. M. Babb, M. A. Leslie, and M. D. Van Syke, "The Potential of Business Gaming Methods in Research," *The Journal of Business*, vol. 39, pp. 465 – 472, 1966.
- [22] G. A. Miller, "The Magical Number Seven, Plus or Minus Two: Some Limits on Our Capacity for Processing Information," *Psychological Review*, vol. 63, pp. 81 – 97, 1956.
- [23] H. A. Simon, *Models of Man Social and Rational: Mathematical Essays on Rational Human Behavior in a Social Setting*. New York: John Wiley & Sons, 1957.
- [24] H. D. Feldman, "Computer-Based Simulation Games: A Viable Educational Technique for Entrepreneurship Classes?," *Simulation & Gaming*, vol. 26, pp. 346 – 360, 1995.
- [25] J. L. McKenney, "An Evaluation of a Business Game in an MBA Curriculum," *The Journal of Business*, vol. 35, pp. 278 – 286, 1962.
- [26] E. Booker, "Have You Driven a Simulated Ford Lately?," in *Computerworld*, vol. 28, 1994, pp. 76.
- [27] G. R. Andlinger, "Looking Around: What Can Business Games Do?," *Harvard Business Review*, vol. 36, pp. 147 – 160, 1958.
- [28] A. Parasuraman, "Assessing the Worth of Business Simulation Games: Problems and Prospects," *Simulation & Games*, vol. 12, pp. 189 – 200, 1981.
- [29] D. L. Meadows, "Gaming to Implement System Dynamics Models," in *Computer-Based Management of Complex Systems*, P. M. Milling and E. O. K. Zahn, Eds. Berlin: Springer-Verlag, 1989, pp. 635 – 640.
- [30] N. Oreskes, K. Shrader-Frechette, and K. Belitz, "Verification, Validation, and Confirmation of Numerical Models in the Earth Sciences," *Science*, vol. 263, pp. 641 – 646, 1994.
- [31] V. Peters, G. Vissers, and G. Heijne, "The Validity of Games," *Simulation & Gaming*, vol. 29, pp. 20 – 30, 1998.
- [32] C. F. Hermann, "Validation Problems in Games and Simulations with Special Reference to Models of International Politics," *Behavioral Science*, vol. 12, pp. 216 – 231, 1967.
- [33] J. S. Goodwin and S. G. Franklin, "The Beer Distribution Game: Using Simulation to Teach Systems Thinking," *Journal of Management Development*, vol. 13, pp. 7 – 15, 1994.
- [34] P. M. Senge, A. Kleiner, C. Roberts, R. B. Ross, and B. J. Smith, *The Fifth Discipline Fieldbook*. London: Nicholas Brealey Publishing, 1994.
- [35] A. Lomi, E. R. Larsen, and A. Ginsberg, "Adaptive Learning in Organizations: A System Dynamics-Based Exploration," *Journal of Management*, vol. 23, pp. 561 – 583, 1997.
- [36] J. D. Sterman, "Modeling Managerial Behavior: Misperceptions of Feedback in a Dynamic Decision Making Environment," *Management Science*, vol. 35, pp. 321 – 339, 1989.
- [37] J. D. Sterman, "Deterministic Chaos in Models of Human Behavior: Methodological Issues and Experimental Results.," *System Dynamics Review*, vol. 4, pp. 148 – 178, 1988.
- [38] E. Mosekilde and E. R. Larsen, "Deterministic Chaos in the Beer Production-Distribution Model," *System Dynamics Review*, vol. 4, pp. 131 – 147, 1988.
- [39] L. A. Belady and M. M. Lehman, "A Model of Large Program Development," *IBM Systems Journal*, vol. 15, pp. 225 – 252, 1976.
- [40] B. W. Boehm, *Software Engineering Economics*. Sydney: Prentice-Hall, 1981.
- [41] T. J. McCabe, "A Software Complexity Measure," *IEEE Transactions on Software Engineering*, vol. 2, pp. 308 – 320, 1976.
- [42] H. Remus and S. Zilles, "Prediction and Management of Program Quality," presented at Proceedings of the 4th International Conference on Software Engineering, Munich, Germany, 1979.
- [43] T. K. Abdel-Hamid and S. E. Madnick, *Software Project Dynamics: An Integrated Approach*. Englewood Cliffs: Prentice-Hall, 1991.
- [44] T. Variaale, B. Rosetta, M. Steffen, H. Rubin, and E. Yourdon, "Modeling the Maintenance Process," *American Programmer*, vol. 7, pp. 29 – 37, 1994.

- [45] G. A. Hansen, "Simulating the Software Development Process," *IEEE Computer*, vol. 29, pp. 73 – 77, 1996.
- [46] J. D. Tvedt, "An Extensible Model for Evaluating the Impact of Process Improvements on Software Development Cycle Time." Phoenix, Arizona: Arizona State University, 1996.
- [47] J. Collofello, "University/Industry Collaboration in Developing a Simulation Based Software Project Management Training Course," presented at Proceedings of the Thirteenth Conference on Software Engineering Education & Training, Austin, Texas, 2000.
- [48] R. J. Madachy, "System Dynamics Modeling of an Inspection-Based Process," presented at Proceedings of the 18th International Conference on Software Engineering, Berlin, Germany, 1996.
- [49] R. Martin and D. M. Raffo, "Application of a Hybrid Process Simulation Model to a Software Development Project," *The Journal of Systems and Software*, vol. 59, pp. 237 – 246, 2001.
- [50] R. Boguslaw and W. J. Pelton, "STEPS, A Management Game for Programming Supervisors," *Datamation*, vol. 5, pp. 13 – 16, 1959.
- [51] J. McCarthy and M. McCarthy, *Software For Your Head*. Boston: Addison-Wesley, 2002.
- [52] R. H. Martin, "A Hybrid Model of the Software Development Process." Portland, Oregon: Portland State University, 2002.
- [53] J. Wolfe, "The Effects of Game Complexity on the Acquisition of Business Policy Knowledge," *Decision Sciences*, vol. 9, pp. 143 – 155, 1978.
- [54] A. P. Raia, "A Study of the Educational Value of Management Games," *The Journal of Business*, vol. 39, pp. 339 – 352, 1966.
- [55] K. E. F. Watt, "Why Won't Anyone Believe Us?," *Simulation*, vol. 28, pp. 1 – 3, 1977.
- [56] R. J. Butler, T. F. Pray, and D. R. Strang, "An Extension of Wolfe's Study of Simulation Game Complexity," *Decision Sciences*, vol. 10, pp. 480 – 486, 1979.
- [57] J. Day, "Software Development as Organizational Conversation: Analogy as a Systems Intervention," *Systems Research and Behavioral Science*, vol. 17, pp. 349 – 358, 2000.
- [58] R. P. Bostrom and J. S. Heinen, "MIS Problems and Failures: A Socio-Technical Perspective. Part I: The Causes," *MIS Quarterly*, vol. 1, pp. 17 – 32, 1977.
- [59] R. P. Bostrom and J. S. Heinen, "MIS Problems and Failures: A Socio-Technical Perspective. Part II: The Application of Socio-Technical Theory," *MIS Quarterly*, vol. 1, pp. 11 – 28, 1977.
- [60] P. D. C. Bennetts, A. T. Wood-Harper, and S. Mills, "The Soft System Methodology as a Framework for Software Process Improvement," *Journal of End User Computing*, vol. 10, pp. 12 – 19, 1998.
- [61] R. Kling and S. Iacono, "The Control of Information Systems Developments After Implementation," *Communications of the ACM*, vol. 27, pp. 1218 – 1226, 1984.
- [62] R. Hirschheim and H. K. Klein, "Four Paradigms of Information Systems Development," *Communications of the ACM*, vol. 32, pp. 1199 – 1216, 1989.
- [63] P. Keen, G. W., "Information Systems and Organizational Change," *Communications of the ACM*, vol. 24, pp. 24 – 33, 1981.
- [64] K. L. Simons, "New Technologies in Simulation Games," *System Dynamics Review*, vol. 9, pp. 135 – 152, 1993.
- [65] J. Wolfe and D. J. Fritzsche, "Teaching Business Ethics with Management and Marketing Games," *Simulation & Gaming*, vol. 29, pp. 44 – 59, 1998.
- [66] P. L. Schumann, P. H. Anderson, and T. W. Scott, "Introducing Ethical Dilemmas into Computer-Based Simulation Exercises to Teach Business Ethics," *Developments in Business Simulations and Experiential Exercises*, vol. 23, pp. 74 – 80, 1996.
- [67] T. DeMarco, *The Deadline: A Novel About Project Management*. New York: Dorset House Publishing, 1997.
- [68] R. J. Dawson, R. W. Newsham, and R. S. Kerridge, "Introducing New Software Engineering Students to the 'Real World' at the GPT Company," *Software Engineering Journal*, vol. 7, pp. 171 – 176, 1992.
- [69] L. Mathiassen, F. Borum, and J. S. Pedersen, "Developing Managerial Skills in IT Organizations—A Case Study Based on Action Learning," *The Journal of Strategic Information Systems*, vol. 8, pp. 209 – 225, 1999.
- [70] R. J. Dawson, R. W. Newsham, and B. W. Fernley, "Bringing the 'Real World' of Software Engineering to University Undergraduate Courses," *IEE Proceedings - Software Engineering*, vol. 144, pp. 287 – 290, 1997.
- [71] J. Wolfe, "A History of Business Teaching Games in English-Speaking and Post-Socialist Countries: The Origination and Diffusion of a Management Education and Development Technology," *Simulation & Gaming*, vol. 24, pp. 446 – 463, 1993.
- [72] H. A. Simon, *The Sciences of the Artificial*, 2nd edition ed. Cambridge, Massachusetts: The MIT Press, 1981.
- [73] J. S. Coleman, "In Defense of Games," in *Gaming-Simulation: Rationale, Design and Applications*, C. S. Greenblat and R. D. Duke, Eds. New York: Sage Publications, 1975, pp. 72 – 74.

Teaching Software Engineering Project Management – A Novel Approach for Software Engineering Programs

Craig Caulfield (Corresponding author)

School of Computer Science and Security Science, Edith Cowan University
2 Bradford Street, Mount Lawley, Western Australia, 6050, Australia
Tel: 61-8-9370-6295 E-mail: ccaulfie@our.ecu.edu.au

David Veal

School of Computer Science and Security Science, Edith Cowan University
2 Bradford Street, Mount Lawley, Western Australia, 6050, Australia
Tel: 61-8-9370-6295 E-mail: d.veal@ecu.edu.au

S. Paul Maj

School of Computer Science and Security Science, Edith Cowan University
2 Bradford Street, Mount Lawley, Western Australia, 6050, Australia
Tel: 61-8-9370-6277 E-mail: p.maj@ecu.edu.au

Received: July 21, 2011

Accepted: August 15, 2011

doi:10.5539/mas.v5n5p87

Abstract

In response to real and perceived short-comings in the quality and productivity of software engineering practices and projects, professionally-endorsed graduate and post-graduate curriculum guides have been developed to meet technical developments and evolving industry demands. Each of these curriculum guidelines identifies better software project management skills as critical for all graduating students, but they provide little guidance on how to achieve this. One possible way is to use a serious game — a game designed to teach and educate players about some of the dynamic complexities of the field in a safe and inexpensive environment. This paper presents the results of a qualitative research project that used a simple game of a software project to see if and how games could contribute to better software project management education. Initial results suggest that suitably-designed games are able to teach software engineering and project management concepts at higher-order Bloom taxonomy levels.

Keywords: Software engineering, Project management education, Peopleware, System dynamics, Serious games

1. Introduction

1.1 Background and Significance

In 1968 and 1969 NATO convened conferences of computer industry representatives and academics to help address what was seen as a growing gap between what was generally hoped for in complex software systems and what was actually achieved (Buxton & Randell, 1970; Naur & Randell, 1969). At the time it was recognised that the demands on software practitioners from industry, defence, and consumers would likely grow at an exponential rate. Yet, software engineering was then more of a craft than a profession (the term software engineering in the conference titles was considered deliberately provocative) and was already struggling to meet quality and performance measures; a software crisis in fact.

By 1982, it was estimated that 15% of all software projects failed to deliver anything, and cost over-runs of 100% to 200% were not uncommon (DeMarco, 1982, p. 3). In the 1990s, little had changed:

For every six new large-scale software systems that are put into operation, two others are cancelled. The average software development project overshoots its schedule by half; larger projects generally do worse. And some three quarters of all large systems are “operating failures” that either do not function as intended or are not used at all.

(Gibbs, 1994, p. 86)

Getting an accurate picture of the current state of the software crisis is difficult because companies are naturally reluctant to publicise failures and they may also oversell their successes. Recent Standish Group CHAOS reports into software project successes and failures (cited in Eveleens & Verhoef, 2010, p. 31) shows an improving trend over the last decade (Table 1), but these reports have been criticised because the research methods and population they are based on are obscure (Eveleens & Verhoef, 2010; Glass, 2006). In the absence of reliable data, it may be conceded that the net societal benefit of software has been positive, but even so the long and expensive history of software project and product failures continues to accrue new examples (see for example Baber, 1982, pp. 26-59; Charette, 2005; Glass, 1998, 1999; Leveson, 1995; Neumann, 1995) and influences how the industry is perceived.

There are some key indicators that the field of software engineering is trying to address these issues. A software engineering body of knowledge (SWEBOK) has been defined to characterise the contents of the software engineering and to provide a foundation for curriculum development (Bourque, Dupuis, Abran, Moore, & Tripp, 1999); there are now professional accreditation and certification programs by which members of the field can be assessed (Naveda & Seidman, 2005); and professionally-endorsed curriculum recommendations have been developed to meet technical developments and evolving industry demands. Of these latter, the following are representative:

- Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering (SE2004) (Joint Task Force on Computing Curriculum, 2004)
- Curriculum Guidelines for Graduate Degree Programs in Software Engineering (GSWE2009) (iSSEc Project, 2009)
- Curriculum Guidelines for Undergraduate Degree Programs in Information Systems (IS2010) (Joint IS2010 Curriculum Task Force, 2010)

Each of these curriculum guidelines mentioned above identifies better software project management skills as critical for all graduating students, but they provide little guidance on how to achieve this. Recognising that competent software engineering students need to supplement the abstract, theoretical side of their studies with some form of practical experience, educational institutions have typically used practicums where the students work in small groups to take a product idea from conception, through design, building and testing, to final delivery. These practicums can be delivered on a number of ways:

- Capstone projects: these are projects designed to synthesise what the students have learned so far and give them a practical way to exercise their skills. The projects themselves may be instructor-designed or proposed by industry and usually cover the final semester of the course (Brereton et al., 2000; Cheng & Lin, 2010).
- Work placements and sandwich courses: students are placed with software companies where they participate in real projects as paid employees. These placements may happen in the later parts of the student's course and may be single opportunities, or intertwined— sandwiched— over a longer period (Lay, Paku, & Swan, 2008; Ribaud & Saliou, 2008).
- Laboratories: student teams work for extended periods on large-scale, ongoing projects within a standardized and evolving development process, which can accommodate team members leaving and joining (Sebern, 2002).

Often, these practicums come near the end of the students' studies, where they can tie together any loose threads by allowing the students to practice what they have learned. "However, this appears to be too little, too late. Projects are often only a single semester in length, students do not benefit from the integration of ideas and practice until the end of their studies, and team orientation is often undermined by scholastic competition for grades" (Schlimmer, Fletcher, & Hermens, 1994).

While the practicums are designed to give students an opportunity to apply their knowledge in a practical way, they often fail because the students are overloaded with many conflicting concerns and often "aren't mature enough to appreciate the importance of many SE topics. On one hand... pay attention to documentation, apply configuration control, test thoroughly... On the other hand, our students have difficulty appreciating issues— such as team organization and cost estimation— that software professionals know from the trenches" (van Vliet, 2006, p. 56).

The purpose of this paper is to explore one way of tackling some of these issues by using a serious game— a game designed to teach and educate players about some of the dynamic complexities software development

projects in a safe and inexpensive environment.

2. Software Engineering Project Management

2.1 Software Project Management in a Social Environment

The sociology of software project management is an often under-represented component in the education and professional development of software engineers even though factors such as team formation, role assignment, motivation, training, hiring, and many other peopleware practices (DeMarco & Lister, 1999) have been identified many times as at least equally important to the success of software projects as the technical (Constantine, 1995; DeMarco, 1991; DeMarco & Lister, 1999; Weinberg, 1998; Yourdon, 1992, 1998, 2004). The reasons for this may be two-fold: the seeming arbitrariness of the sociological factors in software development is at odds with the formal and familiar technical aspects; and the lack of suitable tools with which to model and understand human dynamics.

Successful project management also depends on accepting that in any social environment, such as a software development team, sensible decisions can result in counter-intuitive, and possibly counter-productive, outcomes. Consider, for example, Brooks' Law from Fred Brooks *Mythical Man Month* (Brooks, 1995). The title refers to that fundamental unit of measurement and scheduling, the man-month; a unit that Brooks believes is often misunderstood:

Cost does indeed vary as the product of the number of men and the number of months. Progress does not. Hence the man-month as a unit for measuring the size of a job is a dangerous and deceptive myth. It implies that men and months are interchangeable. (Brooks, 1995, p. 16)

Because of this lack of interchangeability, Brooks' informal law states that adding more developers to a late software project in the hope of meeting a looming deadline will only make matters worse. The reason lies in the fact that software projects often cannot be broken into isolated, independent units of work, meaning that the developers need to coordinate their activities at a detailed level. Therein lies an unappreciated communications overhead. For example, if a group of n developers need to coordinate their efforts with each other then the number of communication paths can be represented by $n(n-1)/2$. Time spent navigating these paths is time not spent being directly productive.

When new developers are added to the equation, the communications overhead is amplified. The new developers are usually not immediately productive because they need to become acquainted with the overall aims of the project, its strategy and the general plan of work (Bradley & McGrath, 2000; Sengupta, Abdel-Hamid, & Bosley, 1999), and they possibly need to undergo some form of organisational socialisation (Schein, 1980). The best, and often only, people able to provide this training and socialisation are the existing developers, who are in the process diverted from their primary tasks.

The net result is that more time is lost in bringing the new developers up to speed and in additional coordination efforts than is gained in productive time (see Caulfield, Kohli, Maj, 2004 for a worked example).

2.2 Software Project Management in the Curriculum

The IS2010 curriculum guidelines address some of these peopleware practices because, "it is impossible for IS graduates to exhibit the required high-level IS capabilities without these foundation knowledge and skills" (Joint IS2010 Curriculum Task Force, 2010, p. 21). The recommended educational experiences include leadership & collaboration; communication, and negotiation. Negotiation skills are needed in order to navigate the often competing interests of the stakeholders involved in a typical project. The recommended course, IS2010.5 IS Project Management, is designed to teach students the processes, methods, techniques, and tools that organizations use to manage their information systems projects. However, "the course specification intentionally leaves discussion regarding specific methods and approaches unanswered" (Joint IS2010 Curriculum Task Force, 2010, p. 50), which means institutions need to figure out for themselves how best to teach these aspects.

Similarly, the SE2004 curriculum guidelines, which are explicitly based on the SWEBOK, specify student outcomes that include:

- Work as an individual and as part of a team to develop and deliver quality software artefacts.
- Reconcile conflicting project objectives, finding acceptable compromises within limitations of cost, time, knowledge, existing systems, and organizations (Joint Task Force on Computing Curriculum, 2004, p. 15).

To achieve these outcomes, the SE2004 guidelines define nine Software Engineering Education Knowledge (SEEK) knowledge areas and associated knowledge units that include Software Management (MGT), which

represents approximately 4% of the taught-load component. For all knowledge areas and units, Bloom (Bloom, Masia, & Krathwohl, 1956) attributes of *knowledge*, *comprehension* or *application* are assigned. To recap, the Bloom taxonomy is a classification of learning objectives consisting of three domains: cognitive, affective and psychomotor. The cognitive domain defines six levels of taxonomy from the lowest to the highest:

- Knowledge: remember previously-learned materials by recalling specific facts, terminology, theories and answers
- Comprehension: demonstrate an understanding of information by being able to compare, contrast, organize, interpret, describe, and extrapolate.
- Application: use previously-learned material in new situations.
- Analysis: decompose previously-learned material into parts in order find patterns and to make inferences and generalizations.
- Synthesis: use existing ideas in different ways to create new ideas or to propose alternative solutions.
- Evaluation: judge the validity of ideas or information with a certain context.

The SE2004 Software Management knowledge area consists of five knowledge units: Management Concepts, Project Planning, Project Personnel and Organization, Project Control and Software Configuration and Management (Table 2). Within this, the knowledge units Project Planning and Project Personnel and Organization are each given the Bloom classification level of *application* (Tables 3, 4). SE2004 curriculum guideline #17 encourages a variety of teaching and learning methods that include problem-based learning, just-in-time learning, learning by failure and self-study. Specifically the Software Project Management course (SE323) identifies sample laboratories and assignments that include:

- Use a commercial project management tool to assist with all aspects of software project management
- Make cost estimates for a small system using a variety of techniques
- Developing a project plan for a significant system
- Writing a configuration management plan
- Using change control and configuration management tools
- Evaluating a software contract or license

In a similar way to IS2010 and SE2004, the GSWE2009 defines a Core Body of Knowledge (CBOK) along with associated Bloom classifications; the distinction between GSWE2009 and SE2004 is that the former takes more units to a higher Bloom taxonomy level:

SE2004 recommends mastery of many topics at level 1. *Every* topic in GSWE2009 must be mastered at level 2 or higher. Moreover, many more topics in GSWE2009 require mastery at level 3 than does SE2004; e.g., in SE2004, the topic of *software process* is addressed only at levels 1 and 2. In GSWE2009, the same topic is covered at levels 2 and 3. (iSSEc Project, 2009, p. 15)

But, software project management is a human-centered activity concerned with a complex and dynamic system often characterised by conflicting demands, changing deadlines, and personality conflicts. It is suggested that these learning outcomes are associated with Bloom taxonomy levels 4, 5 and 6.

3. Simsoft

3.1 Background

In the previous section it was shown that the various software engineering and information systems curriculums place great emphasis on making sure graduates are cognisant of the value of sound software project management, including peopleware, but they provide little guidance on how to achieve this. Given that software development projects are complex socio-technical systems then arguably what is needed is an instructional method that provides students with an opportunity to experience the dynamics of a software project in something akin to a real-world environment. Importantly, this experience needs to demonstrate how a project can rapidly escalate out of control, for example through Brooks' Law, even though seemingly sensible decisions have been made.

But, experience can be expensive. There is a story of a young IBM executive whose innocent mistake caused a \$10 million loss for the company. Coming before Thomas J Watson, the formidable IBM boss, the contrite executive said, "I'm here to tender my resignation". Watson replied, "You must be kidding! We've just spent ten million dollars training you" (Awad & Ghaziri, 2008, p. 281).

The young IBM executive was lucky to have an enlightened boss, but must things always happen this way? Must mistakes be made in the real before we can learn from them? Perhaps not: games are a way of experiencing the real in a controlled and inexpensive way so that software engineers and software project managers don't repeat the same expensive mistakes (cost and time over-runs, dissatisfied end-users, burnt out staff, unstable or unreliable software) that bedevil modern software projects (Caulfield & Maj, 2008; Caulfield, 2002). Of course, games aren't the only way of achieving this, but:

- Games have been used as learning tools in many different business, military, and social environments, and have proven to be efficacious (Gee, 2007a; Michael & Chen, 2005; Perla, 1990; Prensky, 2007; Schrage & Peters, 1999).
- Games draw their intellectual integrity from a number of sources including educational theory (Dewey, 1938/1963; Kolb, 1984; Papert, 1980), operations research (Thomas & Deemer, 1957; Wilson, 1968, pp. 36-50), small-group behaviour research (Kennedy, 1971a, 1971b), war-gaming, decision sciences, and systems engineering (Raser, 1969, pp. 46-55), and problem-based learning (Savin-Baden & Major, 2004).

So, games have a pedigree to be taken seriously as research and learning tools. For this research project, a game called Simsoft (Caulfield, Veal, & Maj, 2011a) was developed to see what contribution it could make to the education of software engineers and software project managers and thereby fill some of the pedagogical gaps in the SE2004, IS2010, and GSWE2009 curriculum guidelines.

3.2 Description of Simsoft

Physically, Simsoft comes in two pieces:

- An A0-sized printed game board around which the players gather to discuss the current state of the project and to consider their next move. The board shows the flow of the game while plastic counters are used to represent the staff of the project. Poker chips represent the team's budget, with which they can purchase more staff, and from which certain game events may draw or reimburse amounts depending on decisions made during the course of the game.
- A simple Java-based dashboard (Caulfield, Veal, & Maj, 2011b) through which the players can:
 - See the current and historical state of the project through a series of simple reports, messages, and other information.
 - Can adjust the project's settings, for example to recruit new staff, before advancing the game's time to create the state of the project.

The aim of the game is to complete the project on time and with funds (poker chips) left over.

The engine behind Simsoft is a model which embodies the fundamental causal relationships of a simple software development project. Software development projects have been popular targets for modellers trying to understand how and why they work the way they do (Abdel-Hamid & Madnick, 1991; Belady & Lehman, 1976; Boehm, 1981; Collofello, 2000; McCabe, 1976; Remus & Zilles, 1979; Tvedt, 1996; Variale, Rosetta, Steffen, Rubin, & Yourdon, 1994). For the research project described here, system dynamics has been used.

System dynamics is a modelling approach to dynamic socio-technical problems, stemming from the work of Forrester (1961, 1969, 1971) at MIT and since developed (Senge, 2006; Sterman, 2000; Wolstenholme, 1990), that allows a modeller to mix soft variables (morale, perceptions, motivations) with familiar hard variables (time, cost, resources). A system dynamics model is not so much a tool for time-point prediction, but more of an experimental device to see how certain variables might change over time under the influence of unappreciated causal relationships, dynamic complexity, and structural delays. The end result is hopefully a more informed mind set with which to manage the situation at hand (C. W. Caulfield & Maj, 2002).

Behind the system dynamics model is be a relational database to store the decisions entered by the players, the parameters which define the particular project (for example, budget and time), and which will capture the state of the model at each time slice. This will allow the game to be rolled backward or forwards, replayed, and studied.

3.3 The Simsoft Game Play

Simsoft players are formed into teams of two or three or more and they are given a scenario that describes the requirements for a small software development project. Taking the role of project manager, the team must manage the project from start-up to final delivery. *What* the players must deliver is handled by boxes on the left side of the Simsoft game board (Figure 1).

At the start of the game there is a pool of work to do. This pool is represented on the game board with small plastic counters in the *Work To Do* box. These counters can be thought of as Use Cases or items in a work breakdown structure; whatever is most familiar to the players. Depending on the resources available to do the work, the units of work (the counters) move from the *Work To Do* box to a *For Review* box, where the work is reviewed before passing to the *Completed Work* box. Not unexpectedly, some work will fail the review and go to the *Rework* box, before passing back to *For Review* and trying again to get to *Completed Work*. The team can reduce the amount of rework by ‘buying’ more quality assurance staff.

The work-to-do, review, rework, work-completed cycle is a fundamental project work structure first discussed and modelled by Roberts (1964). Roberts’ initial work has been expanded greatly by subsequent researchers who have added rich details based on actual projects (see Lyneis & Ford, 2007 for a comprehensive survey of the field), but the underlying work structure remains unchanged.

Based on the starting scenario of the game, information provided during the game, and their own real-world experience, the players make decisions about how to proceed: whether to hire more staff, what hours should be worked and so on. The team is given a budget for the project (poker chips), with which they ‘buy’ more staff. But, there are trade-offs: more experienced (and therefore more productive) staff are more expensive (New Hires are \$500, Quality Assurance are \$600, Mid-Rangers are \$700, and Old Hands are \$1000), and the staff do not become available immediately— there are recruiting delays to be considered (Yourdon, 1998, p. 98). The players can also see from the game board (Figure 2), that staff naturally gain experience (and therefore become more productive) as the project proceeds— something further they need to consider before spending their precious budget chips.

These decisions are entered through the software dashboard (Figure 3), project time is advanced by one week, and the dashboard tells the participants which pieces to move around the board. The game is now in a new state, which the participants must interpret and then consider their next move.

As in the real world, not everything runs smoothly in Simsoft world and the players may need to rethink their plan. At random times, Simsoft will generate one of the following events:

- A major design flaw has been discovered. Add 5 more units of work to the Rework box.
- Your team wins lotto and three staff have resigned, effective immediately. Remove three staff from the game board.
- The Finance department have made a mistake. Collect \$500 from the bank.

Events like these are called games pulses: an event outside of normal play that the teams must take account of when formulating their next decision set (Duke, 1980, p. 368; Schumann, Anderson, & Scott, 1996; Wolfe & Fritzsche, 1998). How the players react to these pulses will be revealed in their subsequent decision sets.

Play continues in this manner until there is no more work to do (all the unit-of-work counters are in the *Completed Work* box of the game board), or until the project deadline passes, whichever comes first. The aim of the game is to deliver the software before the deadline and on budget (with poker chips left over).

4. Evaluation

4.1 Simsoft Game Sessions

For the research project described in this paper, a series of game sessions were conducted between May and September 2010. Purposive sampling (Lincoln & Guba, 1984, p. 40; Patton, 2002) was used to select the participants of the study from the following pools:

- Post-graduate project management students from two Perth, Western Australia universities.
- Software engineers, project managers, and account managers from a Perth-based software consulting company.

Although the participants (n=59) each had an information technology or project management background, they exhibited notable variances in experience (from recent graduates to 25-year industry veterans); skills (from those still studying to highly-certified professionals); and cultural diversity (the participants came from Australia, Europe, the Middle East, Asia, and South Africa).

Simsoft was used as the primary research tool, before and after which players completed a survey. The pre-game survey was designed to assess the players’ knowledge of general software engineering and project management concepts; and the post-game survey was designed to capture their experience of playing the games, whether they found it useful, and how it might compare to other forms of instruction such as lectures or case studies.

Therefore, this research project had multiple data sources: the Simsoft game database, the pre- and post-game surveys, interviews with the players, researcher memos (Maxwell, 2004, p. 12), and field notes.

4.2 Learning-Design Principles in Simsoft

In his seminal book on video games and education, *What Video Games Have to Teach Us About Learning and Literacy*, Gee (2007b) discusses 36 principles of learning he believes should be designed into every good game. Originally conceived for video games, and later condensed to 13 (Gee, 2007a) under three main categories (empowering users, problem solving, and understanding), the principles parallel those found by other cognitive researchers (Bereiter & Scardamalia, 1993; diSessa, 2000) and they have since been adopted for situations involving an active learner and any game. It is instructive to see how Simsoft addresses Gee's principles (Table 5).

In summary, Simsoft addressed Gee's learning principles this way:

- Empowering users: meets the criteria of empowering users allowing them to organize themselves, take on different roles and have full control over their workforce, subject to budget constraints and hiring delays.
- Problem solving: the problem solving aspect of Simsoft allowed students to experience initially a well ordered problem, in particular human resource, which required more complex decisions as the game proceeded. Significantly game players experienced the causal loop that invariably can lead to the counterintuitive outcomes in project cycles. As noted by one participant, 'We have to be careful about bringing on too many new hires. It'll ultimately clog things up.'
- Understanding: experienced software developers indicated the game had demonstrated aspects of systems thinking in which things fit into a larger systems in which they have meaning. This was evident by comments that included: 'Now I see why' and 'I hope that future versions will let me set up specific scenarios and play them out. That would really help me at work'.

A simple game like Simsoft cannot hope to fully address each of the above learning principles and call itself, in Gee's loaded term, a good game, at least in its first iteration. Nevertheless, Simsoft comes close, if not for the tolerable parity demonstrated in Table 5, then only for the final comment against principle 13. A student was seen to scribble on a game board beside the *Rework* box, "I must remember this". If Simsoft's *raison d'être* is to allow software professionals to fail early and often in a place where failure is safe and can be learned from, then this comment shows that at least one person will be carrying a useful nugget of information into their next project.

The results were further analysed in the context of Bloom's (1956) cognitive taxonomy. Of particular interest for this research project was how Simsoft addressed the higher-order Bloom levels of analysis, synthesis and evaluation:

- Analysis: Simsoft provided players with the opportunity to formulate and assess the evaluations of both themselves and other team players. After the game sessions, the players were invited to stay and discuss their results with other teams. Often these post-game gatherings lasted longer the game sessions themselves as the players gathered around the boards and discussed strategies and experiences.
- Synthesis: Simsoft provided students with the opportunity to aggregate the elements of resourcing into a dynamic, interactive whole. For example, one player commented: 'I see my part in the machinery now'.
- Evaluation: Simsoft provided players with the opportunity to analyse the elements of resourcing, their relationships and organizational principles.

On this basis, Simsoft would be a suitable pedagogical tool in curriculums from SE2004 and up to and including IS2010 and GSWE2009.

5. Conclusions

The preliminary results of this research project suggest that Simsoft meets the criteria of the higher-order Bloom taxonomy levels of analysis, synthesis and evaluation and as such could be used as a viable teaching approach by the IS2010 curriculum. Furthermore, Simsoft may be used to teach the dynamic, human-centered aspects of software project management identified in the SE2004 curriculum, for example as a useful laboratory exercise. It is also submitted that Simsoft may be used as the basis of a graduate program such as GSWE2009 to emphasize the topic of software project management and meet the requirement of raising the Bloom taxonomy level.

While Simsoft could be used at many points during these programs, it is at the end, where the students are preparing for their capstone project or work placement assignments— and where the curriculum guides provide little guidance— that it would be of most use. Students enter these final phases often with little preparation for

the realities of working in teams and delivering a real product. Admitted, they may learn by doing and learn from their mistakes, but in doing they risk their academic grades or the time and money of their sponsor. Games such as Simsoft can move this learning-by-doing and learning-through-failure into a safe and inexpensive environment.

References

- Abdel-Hamid, T. K., & Madnick, S. E. (1991). *Software Project Dynamics: An Integrated Approach*. Englewood Cliffs: Prentice-Hall.
- Awad, E. M., & Ghaziri, H. M. (2008). *Knowledge Management*. Delhi: Dorling Kindersley.
- Baber, R. L. (1982). *Software Reflected: The Socially Responsible Programming of Our Computers*. Amsterdam: North-Holland Publishing Company.
- Belady, L. A., & Lehman, M. M. (1976). A Model of Large Program Development. *IBM Systems Journal*, 15(3), 225 – 252. doi:10.1147/sj.153.0225, <http://dx.doi.org/10.1147/sj.153.0225>
- Bereiter, C., & Scardamalia, M. (1993). *Surpassing Ourselves: An Inquiry into the Nature and Implications of Expertise*. Chicago: Open Court.
- Bloom, B. S., Masia, B. B., & Krathwohl, D. R. (1956). *Taxonomy of Educational Objectives: The Classification of Educational Goals* (Handbook I: Cognitive Domain ed.). London: Longman.
- Boehm, B. W. (1981). *Software Engineering Economics*. Sydney: Prentice-Hall.
- Bourque, P., Dupuis, R., Abran, A., Moore, J. W., & Tripp, L. (1999). The Guide to the Software Engineering Body of Knowledge. *IEEE Software*, 16(6), 35 - 44. doi:10.1109/52.805471, <http://dx.doi.org/10.1109/52.805471>
- Bradley, J., & McGrath, G. M. (2000). *Boot Camp or Bordello: Whipping Rookies into Shape*. Proceedings of the Twenty First International Conference on Information Systems, 467 – 472
- Brereton, O. P., Lees, S., Bedson, R., Boldyreff, C., Drummond, S., Layzell, P. J., et al. (2000). Student Group Work Across Universities: A Case Study in Software Engineering. *IEEE Transactions on Education*, 43(4), 394 – 399
- Brooks, F. P. (1995). *The Mythical Man-Month: Essays on Software Engineering* (20th anniversary ed.). Sydney: Addison-Wesley.
- Buxton, J. N., & Randell, B. (Eds.). (1970). *Software Engineering Techniques: Report on a Conference Sponsored by the NATO Science Committee, Rome, Italy, 27th to 31st October 1969*. Brussels: Scientific Affairs Division, NATO
- Caulfield, C. W., & Maj, S. P. (2008). Come Play. In M. Iskander (Ed.), *Innovative Techniques in Instruction Technology, E-learning, E-assessment, and Education* (pp. 86-91). New York: Springer Netherlands.
- Caulfield, C. W. (2002). *A Case for Games in Software Engineering*. Proceedings of the 8th Australian and New Zealand Systems Conference, Mooloolaba, Queensland.
- Caulfield, C. W., & Maj, S. P. (2002). A Case for System Dynamics. *Global Journal of Engineering Education*, 6(1), 25 – 34
- Caulfield, C. W., Kohli, G. and Maj, S. P. (2004). Sociology in Software Engineering. Proceedings of the 2004 American Society for Engineering Education Annual Conference & Exposition (Salt Lake City). American Society for Engineering Education
- Caulfield, C. W., Veal, D., & Maj, S. P. (2011a). Implementing System Dynamics Models in Java. *International Journal of Computer Science and Network Security* 11(7), 43 – 49
- Caulfield, C. W., Veal, D., & Maj, S. P. (2011b). Teaching software engineering management – issues and perspectives. *International Journal of Computer Science and Network Security*, 11(7), 50 – 54
- Charette, R. N. (2005). Why Software Fails. *IEEE Spectrum*, 42(9 (INT)), 36 – 43
- Cheng, Y.-P., & Lin, J. M.-C. (2010). A Constrained and Guided Approach for Managing Software Engineering Course Projects. *IEEE Transactions on Education*, 53(3), 430 – 436
- Collofello, J. (2000). *University/Industry Collaboration in Developing a Simulation Based Software Project Management Training Course*. Paper presented at the Proceedings of the Thirteenth Conference on Software Engineering Education & Training, Austin, Texas.

- Constantine, L. L. (1995). *Constantine on Peopeware*. Englewood Cliffs: Yourdon Press.
- DeMarco, T. (1982). *Controlling Software Projects*. New York: Yourdon Press.
- DeMarco, T. (1991). *Non-Technological Issues in Software Engineering*. Paper presented at the Proceedings of the 13th International Conference on Software Engineering, Austin, Texas.
- DeMarco, T., & Lister, T. (1999). *Peopeware: Productive Projects and Teams* (2nd edition ed.). New York: Dorset House Publishing Co.
- Dewey, J. (1938/1963). *Experience and Education*. New York: Collier Books.
- diSessa, A. A. (2000). *Changing Minds: Computers, Learning, and Literacy*. Cambridge, Massachusetts: The MIT Press.
- Duke, R. D. (1980). A Paradigm for Game Design. *Simulation & Games*, 11(3), 364 – 377. doi:10.1177/104687819903000409 <http://dx.doi.org/10.1177/104687819903000409>
- Eveleens, J. L., & Verhoef, C. (2010). The Rise and Fall of the Chaos Report Figures. *IEEE Software*, 27(1), 30 – 36. doi:10.1109/MS.2009.154, <http://doi.ieeecomputersociety.org/10.1109/MS.2009.154>
- Forrester, J. W. (1961). *Industrial Dynamics*. Waltham: Pegasus Communications.
- Forrester, J. W. (1969). *Urban Dynamics*. Portland: Productivity Press.
- Forrester, J. W. (1971). *World Dynamics*. Portland: Productivity Press.
- Gee, J. P. (2007a). *Good Video Games and Good Learning: Collected Essays on Video Games, Learning and Literacy*. New York: Peter Lang Publishing.
- Gee, J. P. (2007b). *What Video Games Have to Teach Us About Learning and Literacy*. New York: Palgrave MacMillan.
- Gibbs, W. W. (1994). Software's Chronic Crisis. *Scientific American*, 271(3), 86 – 95
- Glass, R. L. (1998). *Software Runaways*. Upper Saddle River: Prentice Hall.
- Glass, R. L. (1999). *Computing Calamities: Lessons Learned from Products, Projects, and Companies That Failed*. Upper Saddle River: Prentice Hall.
- Glass, R. L. (2006). The Standish Report: Does It Really Describe a Software Crisis? *Communications of the ACM*, 49(8), 15 – 16. doi:10.1145/1145287.1145301, <http://dx.doi.org/10.1145/1145287.1145301>
- iSSEc Project. (2009). *Graduate Software Engineering 2009 (GSWE2009): Curriculum Guideline for Graduate Degree Programs in Software Engineering*.
- Joint IS2010 Curriculum Task Force. (2010). *Curriculum Guideline for Undergraduate Degree Programs in Information Systems*: Association for Computing Machinery and Association for Information Systems.
- Joint Task Force on Computing Curriculum. (2004). *Software Engineering 2004: Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering*: IEEE Computer Society/Association for Computing Machinery.
- Kennedy, J. L. (1971a). Simulation Study of Competition in an "Open World". *Journal of Applied Psychology*, 55(1), 42 – 45. doi:10.1037/h0030598 <http://dx.doi.org/10.1037/h0030598>
- Kennedy, J. L. (1971b). The System Approach: A Preliminary Exploratory Study of the Relation Between Team Composition and Financial Performance in Business Games. *Journal of Applied Psychology*, 55(1), 46 – 49. doi:10.1037/h0030599 <http://dx.doi.org/10.1037/h0030599>
- Kolb, D. A. (1984). *Experiential Learning: Experience as the Source of Learning and Development*. Englewood Cliffs: Prentice-Hall.
- Lay, M. C., Paku, L. K., & Swan, J. E. (2008). *Work Placement Reports: Student Perceptions*. 19th Annual Conference of the Australasian Association for Engineering Education: To Industry and Beyond.
- Leveson, N. G. (1995). *Safeware: System Safety and Computers*. Reading: Addison-Wesley Publishing Company.
- Lincoln, Y. S., & Guba, E. G. (1984). *Naturalistic Inquiry*. London: Sage Publications.
- Lyneis, J. M., & Ford, D. N. (2007). System Dynamics Applied to Project Management: A Survey, Assessment, and Directions for Future Research. *System Dynamics Review*, 23(2 – 3), 157 – 189. doi:10.1002/sdr.377, <http://dx.doi.org/10.1002/sdr.377>

- Maxwell, J. A. (2004). *Qualitative Research Design: An Interactive Approach* (2nd edition ed.). Thousand Oaks: Sage Publications.
- McCabe, T. J. (1976). A Software Complexity Measure. *IEEE Transactions on Software Engineering*, 2(4), 308 – 320. doi:10.1109/TSE.1976.233837, <http://doi.ieeecomputersociety.org/10.1109/TSE.1976.233837>
- Michael, D., & Chen, S. (2005). *Serious Games: Games That Educate, Train, and Inform*. Boston: Thomson Course Technology PTR.
- Naur, P., & Randell, B. (Eds.). (1969). *Software Engineering: Report on a Conference Sponsored by the NATO Science Committee, Garmisch, Germany, 7th to 11th October 1968*. Brussels: Scientific Affairs Division, NATO
- Naveda, J. F., & Seidman, S. B. (2005). Professional Certification of Software Engineers: The CSDP Program. *IEEE Software*, 22(5), 73 – 77. doi:10.1109/MS.2005.132, <http://doi.ieeecomputersociety.org/10.1109/MS.2005.132>
- Neumann, P. G. (1995). *Computer-Related Risks*. Reading, Massachusetts: Addison-Wesley Publishing Company.
- Papert, S. (1980). *Mindstorms*. Brighton, Sussex: The Harvester Press.
- Patton, M. Q. (2002). *Qualitative Research and Evaluation Methods* (3rd edition ed.). Thousand Oaks: Sage Publications.
- Perla, P. P. (1990). *The Art of Wargaming: A Guide for Professionals and Hobbyists*. Annapolis, Maryland: Naval Institute Press.
- Prensky, M. (2007). *Digital Game-Based Learning*. St. Paul, Minnesota: Paragon House Publishers.
- Raser, J. R. (1969). *Simulation and Society: An Exploration of Scientific Gaming*. Boston: Allyn and Bacon Inc.
- Remus, H., & Zilles, S. (1979). *Prediction and Management of Program Quality*. Proceedings of the 4th International Conference on Software Engineering, Munich, Germany, 341 – 350
- Ribaud, V., & Saliou, P. (2008). *Evolution of an Integrated Course Towards a Sandwich Course*. ACM-IFIP IEEEIII 2008 Informatics Education Europe III Conference.
- Roberts, E. B. (1964). *The Dynamics of Research and Development*. New York: Harper & Row.
- Savin-Baden, M., & Major, C. H. (2004). *Foundations of Problem-Based Learning*. Maidenhead: The Society for Research into Higher Learning & Open University Press.
- Schein, E. H. (1980). *Organizational Psychology* (3rd edition ed.). Englewood Cliffs: Prentice-Hall.
- Schlimmer, J. C., Fletcher, J. B., & Hermens, L. A. (1994). Team-Oriented Software Practicum. *IEEE Transactions on Education*, 37(2), 212 – 220
- Schrage, M., & Peters, T. (1999). *Serious Play : How the World's Best Companies Simulate to Innovate*: Harvard Business School Press.
- Schumann, P. L., Anderson, P. H., & Scott, T. W. (1996). Introducing Ethical Dilemmas into Computer-Based Simulation Exercises to Teach Business Ethics. *Developments in Business Simulations and Experiential Exercises*, 23, 74 - 80
- Sebern, M. J. (2002). *The Software Development Laboratory: Incorporating Industrial Practice in an Academic Environment*. Proceedings of the 15th Conference on Software Engineering Education and Training, 118
- Senge, P. M. (2006). *The Fifth Discipline: The Art & Practice of The Learning Organization* (Revised edition ed.). London: Random House Business Books.
- Sengupta, K., Abdel-Hamid, T. K., & Bosley, M. (1999). Coping with Staffing Delays in Software Project Management: An Experimental Investigation. *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, 29(1), 77 – 91
- Sterman, J. D. (2000). *Business Dynamics: Systems Thinking and Modelling for a Complex World*. New York: Irwin McGraw-Hill.
- Thomas, C. J., & Deemer, W. L. (1957). The Role of Operational Gaming in Operations Research. *Operations Research*, 5(1), 1 – 27

- Tvedt, J. D. (1996). *An Extensible Model for Evaluating the Impact of Process Improvements on Software Development Cycle Time*. Unpublished Unpublished Ph.D. dissertation, Arizona State University, Phoenix, Arizona.
- van Vliet, H. (2006). Reflections on Software Engineering Education. *IEEE Software*, 23(3), 55 – 61. doi: 10.1109/MS.2006.80, <http://doi.ieeecomputersociety.org/10.1109/MS.2006.80>
- Variale, T., Rosetta, B., Steffen, M., Rubin, H., & Yourdon, E. (1994). Modeling the Maintenance Process. *American Programmer*, 7(3), 29 – 37
- Weinberg, G. M. (1998). *The Psychology of Computer Programming* (silver anniversary edition ed.). New York: Dorset Housing Publishing.
- Wilson, A. (1968). *The Bomb and the Computer*. London: Barrie & Rockliff, The Cresset Press.
- Wolfe, J., & Fritzsche, D. J. (1998). Teaching Business Ethics with Management and Marketing Games. *Simulation & Gaming*, 29(1), 44 – 59. doi:10.1177/1046878198291005 <http://dx.doi.org/10.1177/1046878198291005>
- Wolstenholme, E. F. (1990). *System Enquiry: A System Dynamics Approach*. Brisbane: John Wiley & Sons.
- Yourdon, E. (1992). *Decline and Fall of the American Programmer*. Sydney: Prentice-Hall.
- Yourdon, E. (1998). *Rise and Resurrection of the American Programmer*. Sydney: Prentice-Hall.
- Yourdon, E. (2004). *Death March* (2nd edition ed.). Upper Saddle River: Prentice Hall.

Table 1. Standish CHAOS report benchmarks

Year	Successful (%)	Challenged (%)	Failed (%)
1994	16	53	31
1996	27	33	40
1998	26	46	28
2000	28	49	23
2004	29	53	18
2006	35	46	19
2009	32	44	24

Table 2. SE2004 SEEK knowledge area and units for Software Management

KA/KU	Title	Hours
MGT	Software Management	19
MGT.con	Management concepts	2
MGT.pp	Project planning	6
MGT.per	Project personnel and organization	2
MGT.ctl	Project control	4
MGT.cm	Software configuration management	5

Table 3. SE2004 project planning topics

KA/KU	Topic	Bloom's taxonomy
MGT.pp	Project planning	
MGT.pp.1	Evaluation and planning	Comprehension
MGT.pp.2	Work breakdown structure	Application
MGT.pp.3	Task scheduling	Application
MGT.pp.4	Effort estimation	Application
MGT.pp.5	Resource allocation	Comprehension
MGT.pp.6	Risk management	Application

Table 4. SE2004 project personnel and organization topics

KA/KU	Topic	Bloom's taxonomy
MGT.per	Project personnel and organization	
MGT.per.1	Organizational structures, positions, responsibilities and authority	Knowledge
MGT.per.2	Formal/informal communication	Knowledge
MGT.per.3	Project staffing	Knowledge
MGT.per.4	Personnel training, career development, and evaluation	Knowledge
MGT.per.5	Meeting management	Application
MGT.per.6	Building and motivating teams	Application
MGT.per.7	Conflict resolution	Application

Table 5. Simsoft evaluation against Gee's learning principles

Learning Principle	In Simsoft
I. Empowered Users	
<p>1. Co-design: good learning means that players feel they are active agents (producers) not just passive recipients (consumers).</p> <p>In good games, players feel their actions and decisions– and not just those of the game designer– are co-designing the game world and the experiences they are having. It therefore matters what the player does because this determines a unique path through the game.</p>	<p>The course of game play in Simsoft is completely determined by the decisions the players make. They have full control of their workforce planning (subject to budget and timing restraints) and can increase or reduce hours as required.</p>
<p>2. Customise: different styles of learning work better for different people. People cannot be agents of their own learning if they cannot make decisions about how they learn best. At the same time, they should be able (and encouraged) to try new styles.</p>	<p>Teams can organise themselves any way they wish. Some nominated a lead decision maker or arbiter, usually based on experience, while others were more collaborative and democratic. the game sessions contained enough. the game sessions contained enough time for the</p>

<p>Good games achieve this by naturally accommodating different styles of learning and playing or by allowing the players customise the game play to fit their style.</p>	<p>players to debate their decisions.</p>
<p>3. Identity: deep learning requires an extended commitment and such a commitment is typically created when people take on a new identity they value and in which they become heavily invested.</p> <p>Good games offer players identities in which they can rewardingly invest time and effort. This can be done by offering a character so intriguing that players want to inhabit the avatar and project onto it their own fantasies, desires, and pleasures. Alternatively, games may offer a relatively empty character upon which players can build a deep and consequential life history.</p>	<p>Players take on the role of a project manager– not something so exciting, particularly for experienced project managers. But a Simsoft project manager is unfettered by project politics and has complete control over the project's budget and workforce planning. This comment was from a project manager:</p> <p>“I wish I have [sic] this power at work”</p>
<p>4. Manipulation and distributed knowledge: cognitive research suggests perception and action are deeply interconnected. "Thus, fine-grained action at a distance - for example, when a person is manipulating a robot or watering a garden via a web cam - cause humans to feel as if their bodies and minds have stretched into a new space. More generally, humans feel expanded and empowered when they can manipulate powerful tools in intricate ways that extend their area of expertise."</p> <p>Good games almost always involve action at a (virtual) distance. The more intricately a player can control a character and objects in the game world, the more the player is willing to invest time and effort in the game.</p>	<p>The players had full control over their workforce, subject to budget constraints and hiring delays.</p>
<p>II. Problem Solving</p>	
<p>5. Well-ordered problems: problems in good games are designed so that the early challenges a player faces allow them to form good hypotheses they can use now and later.</p>	<p>Initially players made simple decisions about hiring more staff to ramp up the project. By the time they were confident with the mechanics of this process, the game state would have changed sufficiently so they would then have to make more complex decisions to balance work backlogs, the volume of rework, a looming deadline and reduced funds.</p>
<p>6. Pleasantly frustrating: learning works best when new challenges are pleasantly frustrating, that is at the outer edge of, but within, the player's regime of competence. These challenges feel hard, but doable.</p>	<p>Simsoft demands more careful decisions as the game progresses. For example, the usual response to a large back log of work is to hire more staff, but the hiring delay means there is no immediate effect. A number of</p>

<p>Players also need feedback so even if they fail, they have an idea of what must be done next time.</p>	<p>teams noticed this during the game: "We have to be careful about bringing on too many new hires. It'll ultimately clog things up". For all teams, the causal loop diagram on the back of the project briefing document was used to point out the counterintuitive nature of many project cycles.</p>
<p>7. Cycles of expertise: expertise in any field is created by repeated cycles of practice until the skills become nearly automatic. New skills are gradually added to the practice set and the cycle continues (Bereiter & Scardamalia, 1993). In games, we see this in the different levels a player must move through: there are cycles of extended practice, a test of mastery, then a new challenge which requires further extended practice. In this way the game moves forward at a predictable pace and the player senses achievement at each mastered skill.</p>	<p>More complex decisions need to be made as the game proceeds, but by this time the players will have mastered the mechanics of the game and the delays and counter-intuitive behaviour that are possible. Simsoft logs all game decisions so these can be studied or replayed.</p>
<p>8. Information should be delivered on demand and just in time: humans are not good at using information when it has little context and before they can practically use it. Instead, information is best used when it is given just in time (when it can be used straight away) and on demand (when there is a need to use it).</p>	<p>Each game session was preceded by a short briefing from the researcher about the mechanics of the game and then most sessions were under way within a couple of minutes. Each game schedule contained a causal loop diagram representing the underlying system dynamics model that players could refer to as needed in light the way pieces were moving on the board. The game board itself also shows the major work and personnel flows of the game.</p>
<p>9. Fish tanks: a fish tank can be a simple eco-system containing just a few controlled variables (water, light, food, fish). As such, it can show interactions between the variables that might otherwise be obscured in the real world. In a similar way, games are simplified systems that stress a few key variables and their interactions meaning players are not overwhelmed by the complexity of a whole system.</p>	<p>Simsoft represents a simplified version of a software project: there are no requirements gathering, deployment, or maintenance phases. Instead, the game concentrates on a single, important factor— human resources—without the noise these other phases may have introduced</p>
<p>10. Sandboxes: in games, as in the real world, sandboxes are safe, protected areas where things cannot go too wrong, too quickly and where any affects on the outside environment are minimised.</p> <p>In a good game, a sandbox may be a tutorial, or the first couple of levels may be sandboxed so that decisions made here do not completely spoil the player's chances</p>	<p>Each game session was preceded by a short briefing from the researcher about how to make and enter game decisions. The range of initial decisions available was small so the players were able to see the flow of work over a number if project weeks before making more influential decisions were made.</p>

<p>later in the game.</p>	
<p>11. Skills as strategies: there is a paradox in Principles 7 and 8: players need to practice certain skills in order to master them, but without a sufficient context, this practice may be seen as pointless.</p> <p>In good games, players learn and practice skills in order to accomplish specific things– they are a strategy for accomplishing something first, and of value as skills in themselves second.</p>	<p>The objective of Simsoft is the completion of the project within budget and on time. The skills the players are developing in the game are directly employed to this end.</p>
<p>III. Understanding</p>	
<p>12. Systems thinking: people learn new things (skills, strategies, and ideas) best when they see how these things fit into a larger system in which they have meaning.</p> <p>Good games help players understand how the simplified world of the game fits into a broader context, either of the game or of the real world.</p>	<p>While Simsoft only represents a slice of a real software development project, that slice sends ripples through most other areas of a typical project. This comment was from a software developer with 2 to 5 years experience:</p> <p>“I see my part in the machinery now”</p>
<p>13. Meaning as action image: humans do not usually think in abstract concepts and according to logical principles. Rather, we think through experiences we have had and then create imaginative reconstructions of that experience. To reason about, say, a football game we think about games we have seen and heard about rather than generalities. For humans, words and abstract concepts have their deepest meanings when they are clearly tied to perception and action in the world.</p>	<p>For experienced software developers and project managers, thinking about their work in concrete rather than abstract terms is easy and connections can be made:</p> <p>“Now I see why”</p> <p>“I hope that future versions will let me set up specific scenario and play them out. That would really help me in my work”</p> <p>For students, with less experience to draw on, meaning as action is harder to create. But, there are signs that experience in the game resonates: from a note scribbled on a game board beside the Rework box:</p> <p>“I must remember this”</p>

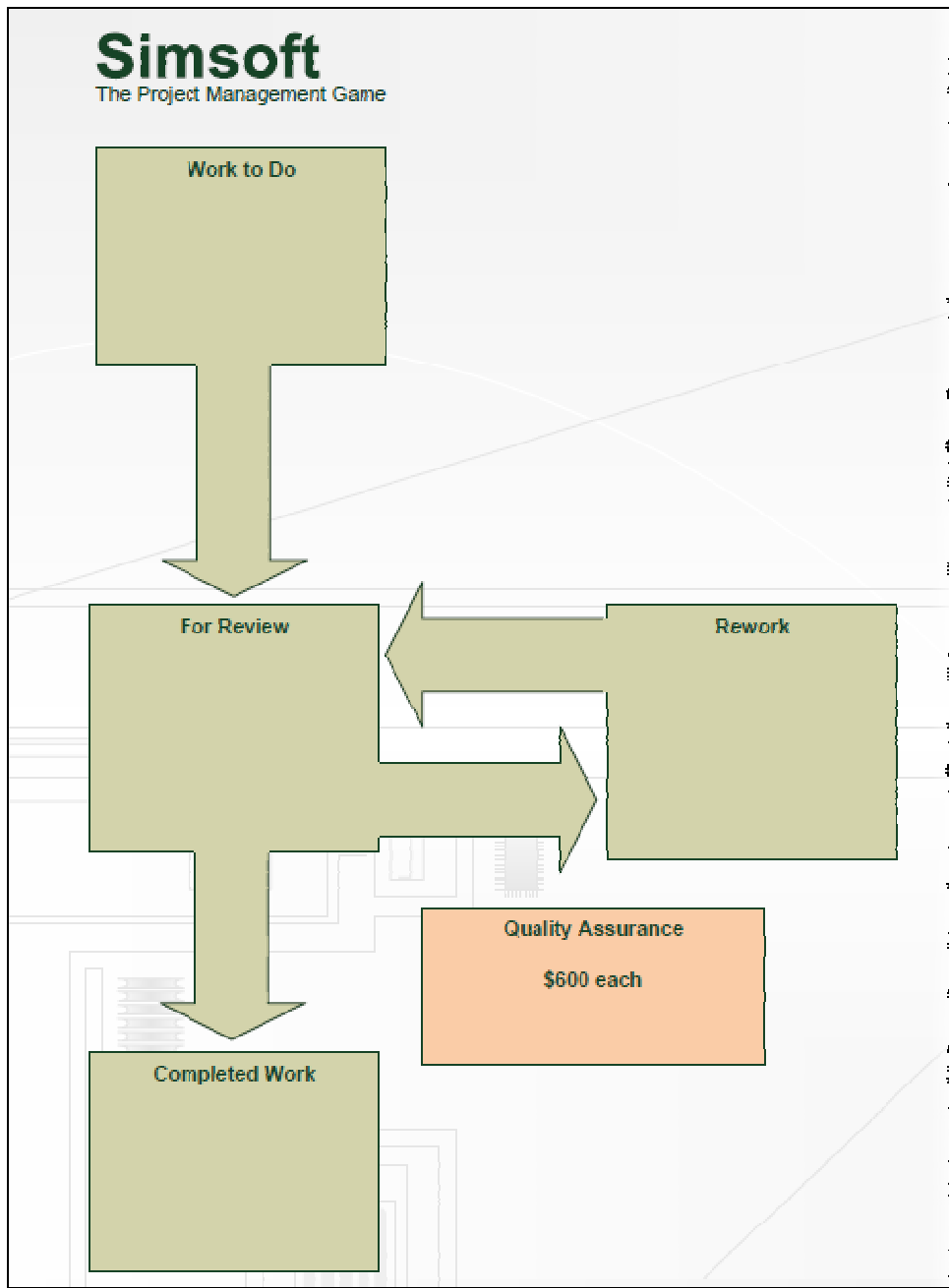


Figure 1. Left-hand side of the Simsoft game board showing the work to be done

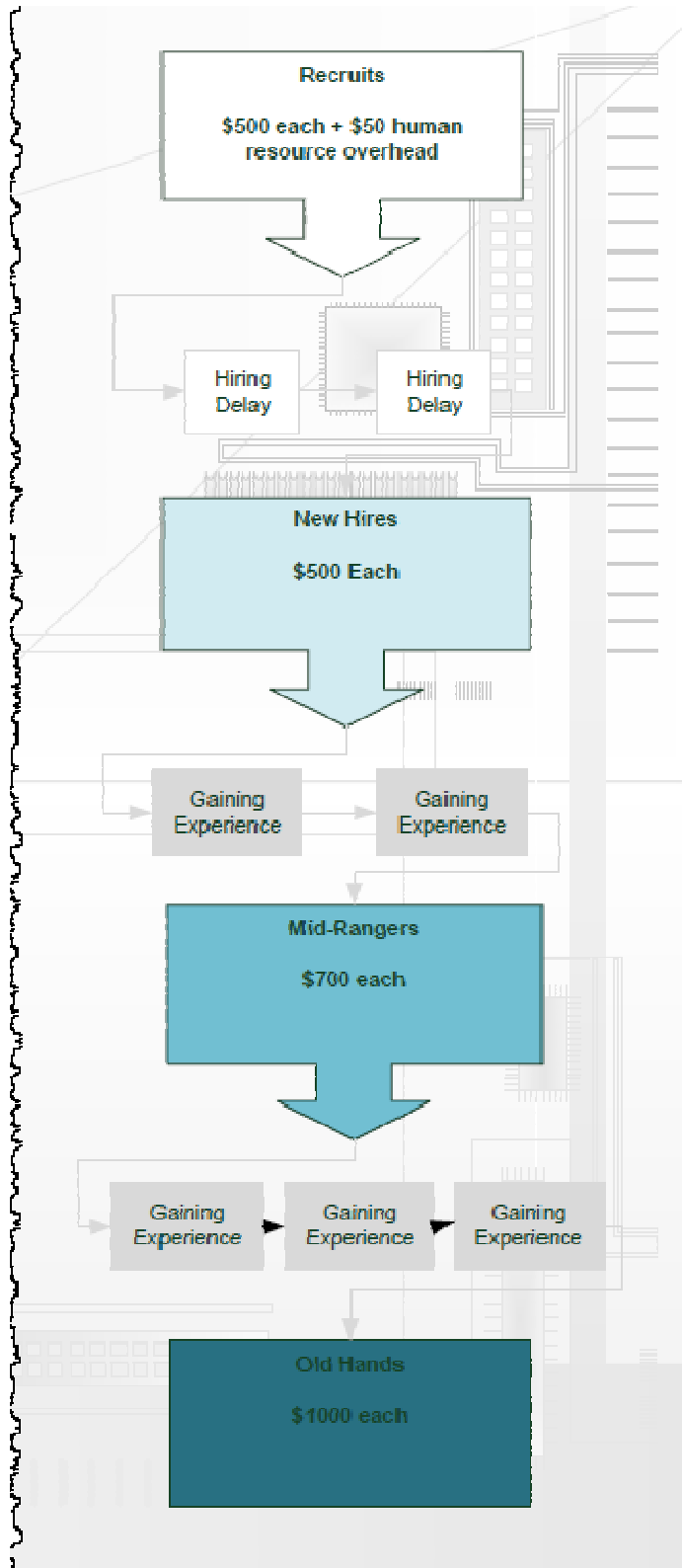


Figure 2. Right-hand side of the Simsoft game board showing the human resources of the project



Figure 3. Simsoft dashboard

Implementing System Dynamics Models in Java

C. Caulfield[†], D. Veal^{††}, S. P. Maj^{†††}

Edith Cowan University, Perth, Western Australia

Summary

For a research project into the value of serious games — games that teach and educate — in software engineering and project management education, a game called Simsoft was developed. Two key parts of Simsoft were the system dynamics engine that captured the fundamental causal relationships of the software project being modelled; and the Java dashboard through which the players entered their project decisions. Java also provided a means of saving the players' individual decisions so these could later be analysed and replayed. While there are currently no Java libraries for implementing system dynamic models, a system dynamics model is simply a collection of non-linear differential equations, and open-source Java libraries for these do exist. Therefore, it is possible to implement a system dynamics model in Java and take advantage of the features of a powerful, general purpose programming language. This paper describes how the model behind Simsoft was created using system dynamics modeling tool called iThink and how the model was subsequently implemented in Java using the Apache Commons Mathematics library.

Key words:

system dynamics, Java, iThink, serious games

1. System Dynamics

1.1 Background and History

In the late 1950s, Jay Forrester of the Sloan School of Management at the Massachusetts Institute of Technology (MIT) was asked by General Electric to review the operations of their Kentucky appliance parts plant. The company was concerned about the oscillating nature of their production cycles that often saw periods of intense activity followed by times of virtual dormancy during which workers had to be laid off. Fluctuating demand and normal business cycles did not seem to adequately explain the situation. Coming from an electrical engineering background and with a keen interest in management science, Forrester approached the problem systematically, but with just a pencil and a note pad. Starting with columns for inventory, employees and orders, and factoring in:

the policies they were following, one could decide how many people would be hired in the following week. This gave a new condition of employment, inventories, and production [1].

Forrester's calculations amounted to a simulation of the system operating at General Electric's plant.

Stemming from this first analysis came an article for the *Harvard Business Review* in 1958 entitled "Industrial Dynamics - A Major Breakthrough for Decision Makers" with the theme being developed and expanded in the seminal work, *Industrial Dynamics* [1, 2]. Industrial dynamics became system dynamics as it came to be used in areas other than industry.

For some time following the publication of *Industrial Dynamics*, system dynamics was used as a tool for looking at big-picture issues such as urban decay, major sociological conditions and world economics [3-5]. In more recent times, system dynamics has been finding a purpose for itself in a range of business and social applications. Instrumental in this change have been Peter Senge's *The Fifth Discipline* [6], and the development of intuitive, graphical software packages that have made system dynamics modelling more accessible by hiding the computer source-code look of traditional models. System dynamics has also found a place for itself in a number of primary, secondary, and tertiary institutions in the United States of America, Australia and Europe, well beyond its ground zero at MIT.

To more formally define system dynamics, it could be said that it:

...is concerned with creating models or representations of real world systems of all kinds and studying their dynamics (or behaviour). In particular, it is concerned with improving (controlling) problematic system behaviour... The purpose in applying System Dynamics is to facilitate understanding of the relationship between the behaviour of the system over time and its underlying structure and strategies/policies/ decision rules [7].

A key element of this definition is the need to build a computer model of the system under consideration. The model is used to help understand the patterns of change or dynamics that a system exhibits over time and to identify the conditions that cause these patterns to be stable or unstable. This knowledge of the system can then suggest what kinds of prescriptions for governing it will work and what kinds may not.

However, building system dynamics models demands persistence. Translating real-world information into model elements is still an inexact science - trial and error can be just as valid as considered judgment based on experience. Perhaps a useful parallel can be drawn with that other hard, inexact activity: finding object-oriented classes. Bjarne Stroustrup, the creator of C++, notes that in design and programming there are no cookbook methods that can replace intelligence, experience and good taste; even he just tries things [8]. The lesson for system dynamics modellers would seem to be the same: just start, try things, take advice of experienced modellers and then keep iterating.

Yet the effort of building a system dynamics model has some benefits:

- Modelling brings about an understanding of the system because of the analytical and critical thinking process it calls for. It helps bring to the surface the mental models driving the current situation - those models

...that one carries around in one's head for dealing with a problem or situation. Such a model maybe based on experience or intuition, or on folklore and myth; it may be influenced by politics and a wide spectrum of human emotions [9]

Mental models may also be totally inappropriate or counter-productive, or equally priceless. But unless they are turned into something more tangible, one may never know.

- System dynamics models make room for both quantitative or hard variables— things that can be measured directly like program size, staffing numbers or dollars spent—; and qualitative or soft variables— such as motivation, commitment, confidence or perceptions. Soft variables have traditionally been left out of engineering models because they are difficult to measure and their importance may have been underestimated. Yet,

...if you omit soft variables you run the risk of failing to capture something essential to driving human affairs. Leaving out something so essential is the only hypothesis that you can reject with absolute certainty! [10].

A system dynamics model can therefore be more informed about its problem space.

With a system dynamics model in hand and George Box's tongue-in-cheek caution in mind (all models are wrong, but some are useful), the model can be run. Certain variables can be held steady while others are changed, it can be placed under stress and tested for sensitivities and leverage points. In short, the model can be experimented with to better understand the present situation and to search for alternatives for improvement. It has been stated that:

The alternatives may come from intuitive insights generated during the [initial analysis], from experience of the analyst, from proposals advanced by people in the operating system [or in the] experience, art, and skill for imagining the most creative and powerful policy alternatives [11].

Peter Senge points out that the causes of many problems

...lay in the very well-intentioned policies designed to alleviate them. These problems were actually systems that lured policy makers into interventions that focused on obvious symptoms not underlying causes, which produced short-term benefit but long term malaise, and fostered the need for still more symptomatic interventions [12].

By simulating a problem space using a system dynamics model, it is possible to potentially make more informed decisions about events beyond our bounded rationality safe from the dangers of real-world experimentation.

1.2 Stock and Flow Diagrams

At its lowest level, a system dynamics model looks like computer source code, but even from the earliest days there were graphical representations to help modellers visualise their problem space. The stock-and-flow notation (Fig. 1), first described by Forrester [1], consists of a small number of symbols that together form a grammar telling a story:

- Stocks or levels can be thought of as nouns since they represent an accumulation of something (money, inventory, staff, morale, etc.) at a point in time.
- Flows or rates determine how the stocks will be filled or drained and so are analogous to verbs. Stuff (again money, inventory, staff, morale, etc.) flows through the pipe of the flow in the direction of the arrow and at a rate determined by the flow regulator in the middle. The flow regulator is fitted with a spigot that can be conceptually tightened or loosened by other variables within the model. The cloud at the end of the flow represents the boundary of the model.
- Converters modify flows within the system, just as adverbs modify verbs. They are often used to break

out the detail of the logic that might otherwise be buried within a flow and might be used to represent constant values. Converters typically influence the behaviours of the regulators on the flows.

- Connectors tie the other three building blocks together. They represent inputs and outputs, not inflows and outflows. Connectors do not take on numerical values—they merely transmit values taken on by other building blocks.

Behind these symbols are stored the functions and values (the 'source code' of the model) that drive the simulation and ultimately produce the output. For a system dynamics model, the output is a multi-scale graph (see Fig. 4 later) that shows how certain variables of interest change over time and in relation to each other.

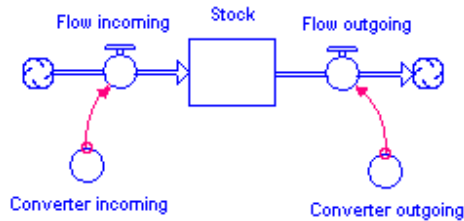


Fig. 1: Stock and flow format of system dynamics models.

2. Basic Mathematics of System Dynamics

The basic mathematics of a system dynamics model is a set of coupled non-linear first-order differential equations [1, 13]. The advance of time is broken into small intervals of equal length (typically called delta time or DT), which is small enough that we can assume change will be constant over that period. For each DT, the model's stocks, rates, converters and auxiliary variables are evaluated to yield a new value, and this value is used as input for continuing calculations.

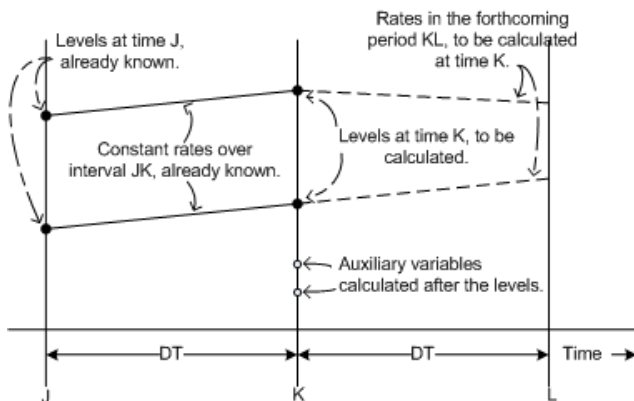


Fig. 2: Calculations at time K

In Fig. 2 [1], J, K, and L represent successive points in time with K being the present. Stock equations are evaluated first and the values are then available for use in the rate equations. Using the simple stock-and-flow diagram in Fig. 1:

$$Stock = \int_0^t (Flow\ incoming - Flow\ outgoing) dt \quad (1)$$

That is, the present value of *Stock* at time *K* is equal to the value of *Stock* at time *J*, plus the difference between the inflow rate and the outflow rate, multiplied by *DT*.

Flows or rates determine how stocks are filled or depleted. To cater for the delay characteristics of information-feedback systems, a rate equation is given by the outflow rate of a first-order exponential delay. For example:

$$Stock\ at\ time\ L = \frac{Stock\ at\ time\ K}{Converter\ outgoing} \quad (2)$$

There are many other specialised functions available to system dynamic modellers, but those for stocks and rates represent the majority of most models.

3. System Dynamics and Java

3.1 Simsoft

For a research project into the value of serious games as teaching tools for software engineers and software project managers, a game — Simsoft — was developed that had as its engine a system dynamics model. Physically, Simsoft comes in two pieces:

- An A0-sized printed game board around which the players gather to discuss the current state of the project and to consider their next move. The board shows the flow of the game while plastic counters are used to represent the staff of the project. Poker chips represent the team's budget, with which they can purchase more staff, and from which certain game events may draw or reimburse amounts depending on decisions made during the course of the game.
- A simple Java-based dashboard through which the players can see the current and historical state of the project through a series of simple reports, messages, and other information; and can adjust the project's settings, for example to recruit new staff, before advancing the game's time to create the state of the project.

The aim of the game is to complete the project on time and with funds (poker chips) left over. At the start of the game there is a pool of work to do. This pool is represented on

the game board with small plastic counters in the *Work To Do* box. These counters can be thought of as Use Cases or items in a work breakdown structure; whatever is most familiar to the players. Depending on the resources available to do the work, the units of work (the counters) move from the *Work To Do* box to a *For Review* box, where the work is reviewed before passing to the *Completed Work* box. Not unexpectedly, some work will fail the review and go to the *Rework* box, before passing back to *For Review* and trying again to get to *Completed Work*.

The work-to-do, review, rework, work-completed cycle is a fundamental project work structure first discussed and modelled by Roberts [14]. Roberts' initial work has been expanded greatly by subsequent researchers who have added rich details based on actual projects (see [15] for a comprehensive survey of the field), but the underlying work structure remains unchanged.

Of interest here is the Java dashboard and the system dynamics model that implements the work-to-do, review, rework, work-completed cycle. The original design of these two components called for a simple but attractive graphical user interface on top of the stock-and-flow plumbing of the system dynamics model; and a means of capturing the decisions made by the players for later analysis. While there are a number of software packages that can create a graphical user interface for system dynamics models [16-18], some problems were encountered:

- There were limited features for creating attractive, interactive user interfaces.
- All packages required some sort of proprietary software to run the model.
- None provided a means to save the individual decisions of multiple teams in a single database so that the decisions could be later analysed or replayed.
- There is a .NET software development kit allows system dynamics models to be integrated with custom-designed software, but this limits further development and deployment to Windows PCs, plus the initial purchase cost and ongoing licensing fees were relatively expensive.

Java was chosen because it addressed each of the above problems. Even so, there are currently no Java libraries for implementing system dynamic models. But, a system dynamics model is simply a collection of non-linear differential equations, and open-source Java libraries for these do exist, therefore it is possible to implement a system dynamics model in Java.

3.2 Model Design in iThink

Building a system dynamics model by hand-coding equations is time-consuming and error prone. Therefore, the model behind Simsoft was first built and tested using a graphical modelling package called iThink [16]. The final model included almost a hundred stocks, flows and their associated equations, so the aim here is to focus on a small part of the model— that of worker burnout as described by Homer [19].

Burnout begins when a person working on a project (in the case of Simsoft, a software engineer on a development project) tries to meet unmet expectations by working longer hours. By working longer hours they are exposed to more of the normal stress of work and consequently their finite store of “adaptive energy” [20, 21] is depleted more quickly and they also have less time to recover. This depleted energy level may leave the person even less capable of meeting their expectations, or may cause them to make mistakes that have to be fixed at the expense of real progress. In response, they may try to work harder, which will deplete their energy levels still more. Unless the person is granted some respite, this viscous cycle may continue until they are leave in frustration or are they burned out and no longer able to contribute to the project.

Fig. 3 shows how burnout can be modelled in iThink.

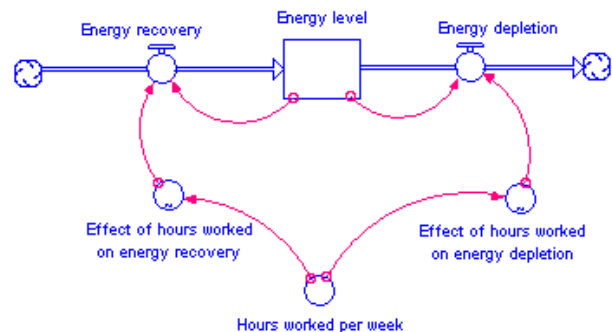


Fig.3: Worker burnout modelled with iThink

Here, a person has a stock of energy available to do work that is depleted or recovered depending on the number of hours they work each week. The effect of hours worked each week on energy recovery and depletion are given in Table 1.

The recovery and depletion rates are nominal values normalised around a 40-hour week. As the number of hours worked each week beyond this point increases, the depletion rate increases; because the person is working

longer days, evenings, or even weekends, there is less time to recover, so the recovery rate slows.

Table 1: Effect of hours worked on energy recovery and depletion

Hours per week	Recovery rate	Depletion rate
0	1.30	0.30
20	1.20	0.60
40	1.10	1.00
60	0.70	1.50
80	0.50	2.00
100	0.35	2.50
120	0.25	3.00

It should be noted that the recovery and depletion rates are known as soft or qualitative variables because they are not based on precise numerical data; such data does not exist [19]. However, the compass of a system dynamics model means that the rules by which it is calibrated and validated will be slightly different from other modelling techniques. For example, the output of a system dynamics model is meant to be read, not for particular time-point predictions, but for qualitative behavioural patterns such as growth, decline, oscillation, stability, and instability [22]. This goal of understanding general dynamic tendencies means that the model's parameters are less reliant on highly precise numerical data:

As long as the purpose of your model is not to predict the numerical magnitude of particular soft variables, you can greatly benefit from including them in your models. Doing so will cause you to think in a rigorous manner about the relationships the variables bear to other variables in the system.[10]

The calibration of soft variables may also seem an arbitrary process in which the model is 'made' to respond in a certain manner. However, the way in which the soft (and hard) variables react must be internally consistent, that is, they must generate behaviour that matches what is observed in the actual system [10].

With this in mind, when this burnout model is run, a multi-scale graph is produced (Fig. 4).

Viewed over a 13-month period, the person starts out by working a 40-hour week. Every couple of weeks there is a spike and they have to work 50-hour weeks for a short time (this pattern can, of course, be changed to model any real-world circumstance). The graph shows that the person's energy levels rise and fall in line with oscillations in the work week, but the overall trend is downwards because the constant spikes in work never allow enough time for proper recovery.

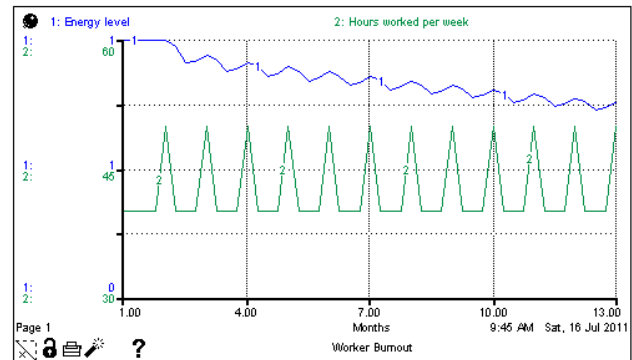


Fig. 4: Worker burnout over a 13-month period

With this portion of the model defined, it only remained to implement it in Java.

3.3 Implementation in Java

The model behind Simsoft was implemented in Java by using the open source Apache Commons Mathematics library [23]. Among its many functions, this library provides a programming interface for solving differential equations.

To implement the system dynamics model shown in Fig. 3, we need to:

- Create a class, *EnergyEquations*, that implements the *FirstOrderDifferentialEquations* interface..
- Pass the class to an integrator to calculate values at different time points. The Apache Commons Mathematics library provides a range of integrator, but for system dynamics models, the Euler or Runge-Kutta methods are most often used.

The key method in *EnergyEquations* is *computeDerivatives*—the one that evaluates the stock equation given at (1).

```
/**
 * Get the current time derivative of the state vector.
 *
 * @param t current value of the independent time
 * variable
 * @param y array containing the current value of the
 * state vector
 * @param yDot placeholder array where to put the time
 * derivative of the state vector
 */
public void computeDerivatives(double t, double[] y,
double[] yDot) throws DerivativeException {
    yDot[0] = y[0] * (recoveryRate - depletionRate);
}
```

EnergyEquations is covered by the following unit test:

```

public void testEnergyEquationsDerivatives() throws
Exception {

FirstOrderIntegrator integrator = new
EulerIntegrator(0.25);
FirstOrderDifferentialEquations energy = new
EnergyEquations(0.7, 1.50);

StepHandler stepHandler = new StepHandler() {
public void handleStep(StepInterpolator interpolator,
boolean isLast) throws DerivativeException {
double t = interpolator.getCurrentTime();
double[] y = interpolator.getInterpolatedState();
System.out.println(t + "\t" + y[0]);
}
public boolean requiresDenseOutput() {return false;}
};
integrator.setStepHandler(stepHandler);
integrator.integrate(energy,
0.75, // start time
new double[]{1.0}, // initial stock value
13, // end time
new double[1]); // storage
}

```

First, an Euler integrator is defined with a step size of 0.25. Then, the EnergyEquations class is constructed and initialised with recovery and depletion rates of 0.7 and 1.50 respectively, being values from Table 1 that equate to a 60-hour week. (An inner StepHandler class is created so we can see the output at each step). Finally, the EnergyEquations instance is passed to the integrator along with the initial conditions of the run. The first data items are shown Table 2.

Table 2: Initial data items from

Time	Energy Level Value
1.00	0.80
1.25	0.64
1.50	0.512
1.75	0.4096
2.00	0.32768
2.25	0.26214400000000004
...	

In essence, the same pattern can be followed for all stocks.

4. Conclusions

System dynamics is concerned with building quantitative and qualitative models of complex problem situations and then experimenting with and studying the behaviour of these models over time. Often such models will demonstrate how unappreciated causal relationships, dynamic complexity, and structural delays may lead to counter-intuitive outcomes of less-informed efforts to improve the situation. System dynamic models also make room for soft factors such as burnout so that problem spaces can ultimately be better understood and managed.

These features made system dynamics an obvious choice for creating the model behind Simsoft because the game was trying to demonstrate some of the dynamic complexities of software development projects. However, the means for implementing system dynamic models and integrating them with custom-designed graphical user interfaces and databases are limited.

By using simple open source tools, such as the Apache Commons Mathematics library, it is possible to build system dynamics models that integrate with general purpose programming languages such as Java, meaning the models can draw upon all the features of those languages. For now this integration is largely manual: create the system dynamics model using tools such as iThink and then translate this into a matching class structure in Java. Based on the results presented here, further research is being conducted into ways of automating this translation and being able to perform round-trip translations.

References

- [1] J.W. Forrester, Industrial Dynamics, Pegasus Communications, Waltham, 1961.
- [2] J.W. Forrester, Harvard Business Review, 36 (1958) 37 - 66.
- [3] J.W. Forrester, Urban Dynamics, Productivity Press, Portland, 1969.
- [4] J.W. Forrester, World Dynamics, Productivity Press, Portland, 1971.
- [5] D.H. Meadows, D.L. Meadows, J. Randers, W.W. Behrens, The Limits to Growth: A Report for the Club of Rome's Project on the Predicament of Mankind, Earth Island Ltd, London, 1972.
- [6] P.M. Senge, The Fifth Discipline: The Art & Practice of The Learning Organization, Revised edition ed., Random House Business Books, London, 2006.
- [7] E.F. Wolstenholme, System Enquiry: A System Dynamics Approach, John Wiley & Sons, Brisbane, 1990.
- [8] B. Stroustrup, The C++ Programming Language, special edition ed., Addison-Wesley, Boston, 2000.
- [9] E. Yourdon, Rise and Resurrection of the American Programmer, Prentice-Hall, Sydney, 1998.
- [10] B. Richmond, Modelling "Soft" Variables, in: An Introduction to Systems Thinking, High Performance Systems, Hanover, 1999, pp. 9-1 - 9-10.
- [11] J.W. Forrester, System Dynamics Review, 10 (1994) 245 - 256.
- [12] P.M. Senge, The Fifth Discipline: The Art & Practice of The Learning Organization, Random House, Milsons Point, 1990.
- [13] A. Ford, Modeling the Environment: An Introduction to System Dynamics Modeling of Environmental Systems, Island Press, Washington, 1999.
- [14] E.B. Roberts, The Dynamics of Research and Development, Harper & Row, New York, 1964.
- [15] J.M. Lyneis, D.N. Ford, System Dynamics Review, 23 (2007) 157 - 189.
- [16] ise Systems (<http://www.iseesystems.com/>), 2011. iThink version 9.1.4.

- [17] Ventana Systems (<http://www.vensim.com/>), 2011. Vensim version 5.
- [18] Powersim (<http://www.powersim.com/>), 2011. Powersim version 8.
- [19] J.B. Homer, System Dynamics Review, 1 (1985) 42 - 62.
- [20] H. Selye, Stress Without Distress, Signet Books, Philadelphia, 1974.
- [21] H. Selye, The Stress of Life, 2nd edition ed., McGraw-Hill, New York, 1978.
- [22] D.H. Meadows, J.M. Robinson, The Electronic Oracle: Computer Models and Social Decisions, John Wiley & Sons, New York, 1985.
- [23] Apache Commons Mathematics Library (<http://commons.apache.org/math/>), 2011. Version 2.2.



Craig Caulfield is a senior software engineer for a technology consulting company and PhD candidate at Edith Cowan University. His research areas include problem-based learning and the application of serious games to software engineering education and project planning.



Dr. David Veal is a Senior Lecturer at Edith Cowan University. He is the manager of Cisco Network Academy Program at Edith Cowan University and be a unit coordinator of all Cisco network technology units. His research interests are in Graphical User Interface for the visually handicapped and also computer network modeling.



A/Prof S. P. Maj has been highly successful in linking applied research with curriculum development. In 2000 he was nominated ECU University Research Leader of the Year award He was awarded an ECU Vice-Chancellor's Excellence in Teaching Award in 2002, and again in 2009. He received a National Carrick Citation in 2006 for "the development of world class curriculum and the design and implementation of associated world-class network teaching laboratories".

Teaching Software Engineering Management – Issues and Perspectives

C. Caulfield, D. Veal, S. P. Maj

Edith Cowan University, Perth, Western Australia

Summary

The ACM/IEEE regularly proposes guidelines for software engineering education, in particular what should be part of the software engineering core body of knowledge and how this knowledge can be taught. The 2004 curriculum guidelines define seven student outcomes, two of which relate to teamwork and project control, and one Software Engineering Education Knowledge (SEEK) area on software management. The software management knowledge area is concerned with the entire software development life cycle and hence the control of people and processes. Significantly, the majority of topics within this area are classified with the Bloom taxonomy level of Application i.e. ability to use learned material in new and concrete situations. However the laboratory and assignment exemplars fail to demonstrate the dynamic, human centered complexity of project management. Simsoft, a serious game, has been designed to potentially address this pedagogical gap.

Key words:

software engineering curriculum, serious games, project management, problem-based learning

1. Software Engineering Curriculum 2004

The Joint Task Force on Computing Curricula (IEEE Computer Society and Association of Computing Machinery) suggests curriculum guidelines for undergraduate degree programs in software engineering. The SE2004 [1] volume defines a core body of knowledge called Software Engineering Education Knowledge (SEEK) which was the basis of curriculum recommendations. SE2004 also defined seven student outcomes that include:

- Work as an individual and as part of a team to develop and deliver quality software artifacts.
- Reconcile conflicting project objectives, finding acceptable compromises within limitations of cost, time, knowledge, existing systems and organizations.

There are ten SEEK knowledge areas— sub-disciplines of the field that undergraduates should know— which are broken down into smaller knowledge units— thematic modules— and finally into topics. Within the Software

Management knowledge area, there are five knowledge units (Table 1).

Table 1: Software management knowledge units

KA/KU	Software Management	Hours Required
MGT.con	Management concepts	2
MGT.pp	Project planning	6
MGT.per	Project personnel and organization	2
MGT.ctl	Project control	4
MGT.cm	Software configuration management	5

Drilling down further, the Project Planning knowledge unit consists of six topics (Table 2), three of which are classified as the Bloom [2] taxonomy level of Application.

Table 2: Project planning topics

Project Planning	Bloom's Taxonomy
Evaluation and planning	Comprehension
Work breakdown structure	Application
Task scheduling	Application
Effort estimation	Application
Resource allocation	Comprehension
Risk management	Application

The Bloom taxonomy is a classification of learning objectives (learning outcomes) consisting of three domains: cognitive, affective and psychomotor. The cognitive domain defines six levels of taxonomy from the lowest to the highest:

1. Knowledge: remember previously-learned materials by recalling specific facts, terminology, theories and answers
2. Comprehension: demonstrate an understanding of information by being able to compare, contrast, organize, interpret, describe, and extrapolate.
3. Application: use previously-learned material in new situations.
4. Analysis: decompose previously-learned material into parts in order find patterns and to make inferences and generalizations.
5. Synthesis: use existing ideas in different ways to create new ideas or to propose alternative solutions.

6. Evaluation: judge the validity of ideas or information with a certain context.

Meanwhile, the Project Personnel and Organization knowledge unit consists of seven topics, three of which are classified as the Bloom taxonomy level of Application (Table 3).

Table 3: Project personnel and organization topics

Project Personnel and Organization	Bloom's Taxonomy
Organizational structures, positions, responsibilities and authority	Knowledge
Formal/informal communication	Knowledge
Project staffing	Knowledge
Personnel training, career development, and evaluation	Knowledge
Meeting management	Application
Building and motivating teams	Application
Conflict resolution	Application

2. SE2004 Courses

The SE2004 curriculum guidelines define topic implementation as a series of courses. Within the context of software engineering management there are three associated courses:

- SE322 Software Requirements Analysis
- SE323 Software Project Management
- SE324 Software Process and Management

The Software Requirements Analysis course is primarily concerned with requirements analysis and modeling. The sample laboratories and assignments require students to use different analysis and modeling tools.

The Software Project Management course is designed to teach project planning. The laboratories and assignments include:

- Use a commercial project management tool to assist with all aspects of software project management
- Make cost estimates for a small system using a variety of techniques
- Developing a project plan for a significant system
- Writing a configuration management plan
- Using change control and configuration management tools
- Evaluating a software contract or license

Furthermore, this unit recommends case studies of real industrial projects.

The Software Process and Management course teaches standards, implementation and assurance of software

processes. No sample laboratories and assignments are provided.

SE2004 curriculum guideline encourages a variety of teaching and learning approaches that include: problem-based learning; just-in-time learning; learning by failure and self-study materials (see for example [3-5]). However in a commercial environment software project management is a human-centered activity that attempts to address the dynamic interactions of factors such as cost, time, staffing, performance, feature set, and quality. A relatively small change in one factor, such as the resignation of a single software engineer, is likely to have a significant impact on the entire project. Whilst learning-to-fail is instructive [6], in a commercial context there are obvious economic implications.

In order to address these concerns, SE2004 includes a capstone project. The course SE400 Software Engineering Capstone Project recommends the development of a significant software system along with all the appropriate artifacts such as project plan, requirements, design documents, test plans etc. Additional teaching considerations include:

- It is suggested that students be required to have a 'customer' for whom they are developing their software
- It is strongly suggested that students work in groups of at least two, and preferably three or four, on their capstone project. Strategies must be developed to handle situations where the contribution of team members is unequal.

3. Meeting a Pedagogical Gap

The authors submit that there is a pedagogical gap between teaching software project management by means of the listed laboratories and assignments and the final capstone project. What is needed is for students to experience and experiment with a dynamic, interactive system that can be deployed by means of, for example, a game. As used here, to play a game:

...is to engage in activity directed towards bringing about a specific state of affairs, using only means permitted by specific rules, where the means permitted by the rules are more limited in scope than they would be in the absence of the rules and where the sole reason for accepting such limitation is to make possible such activity.[7]

The type of game that this paper is concerned with uses an adjective—serious—to show they want for more than simple amusement and that they are designed to educate, train, or inform their players [8-10].

A suitably designed game can not only can mimic real world complexity but can also provide immediate feedback regarding system performance and the effect of decisions made [11].

For example, in an idealized learning process (Figure 1), we receive information in its many forms from the real world in which we live, yet this information can be incomplete, biased, delayed, or in other ways distorted. Still, based on this information, we make decisions that are in turn filtered through our existing mental models, in the process changing or confirming the structure of our real-world systems and creating new decision rules and new strategies or reinforcing the existing. The process then repeats against this new baseline. Games act as an alternative to applying our decisions to the real-world, a way of quickly, inexpensively, and consistently experimenting with different ideas and thereby increasing our store of contexts.

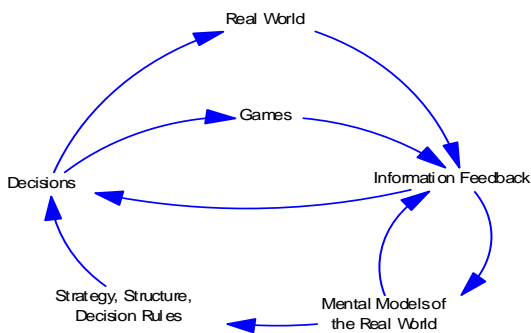


Fig.1: Idealised learning process.

For example, at its simplest, a game, such as the Beer Game, a four-point distribution game developed originally at MIT, can be used to show the cascading effects of a single compensating decision [12-14]. When using the beer game to teach planning, Caulfield [15] found that:

The participants reported a sense of having little control over their ordering decisions and tended to see the root cause of their inventory problems as being caused by other points in the supply chain.

Results such as this can potentially improve learning outcomes because the players can see the results of their actions and have to react accordingly.

To achieve this effect, games do not need high fidelity and need not be overly complex. In fact, it has been demonstrated that whilst:

...the most complex game offered the richest learning experience available, the game's very formidable appearance probably intimidated a number of players or

faced them into a learning situation they were unprepared or unwilling to negotiate [16]

That is, rich and complex games can be daunting for players and they may not be willing to devote the time and effort to play it in depth. The next most effective game in Wolfe's study was found to be the least complex, supporting similar research that showed relatively simple games can provide essentially the same benefits as the more complex [17-19]. Game design is therefore of paramount importance.

4. Simsoft

This paper reports the initial findings of a research project that developed a game called Simsoft to teach software project management. A series of game sessions were conducted with teams of post-graduate project management students (for software and general projects), and practising software project managers and developers (n=59) between May and September 2010. The data sources for the findings were the participants' performance in Simsoft, pre- and post-game surveys, interviews with the participants, and a qualitative rich analysis of the interactions that were observed during the game sessions.

Physically, Simsoft comes in two pieces. There is an A0-sized printed game board around which the players gather to discuss the current state of a project and to consider their next move. The board shows the flow of the game while plastic counters are used to represent the staff of the project. Poker chips represent the team's budget, with which they can purchase more staff, and from which certain game events may draw or reimburse amounts depending on decisions made during the course of the game.

There is also a simple Java-based dashboard, through which the players can see the current and historical state of the project through a series of simple reports, messages, and other information; and can adjust the project's settings, for example to recruit new staff, before advancing the game's time to create the state of the project.

The aim of the game was to complete the project on time and with funds (poker chips) left over.

4.1 Simple Versus Complex Games

The players' responses to different features of the game were generally positive (Table 4). Notable in Table 4 is that a majority of players (44 out of 59) preferred playing with a game board rather than a fully computerized version. Some typical comments were:

"The board game [was] simple and I could easily see the state of the game"

“When a group plays the game on a PC, someone controls the mouse and keyboard and they tend to dominate”
“Compared to computer-based games, the design was simple and we started playing without too much wasted time”
“Sometimes technology gets in the way”
“Everyone plays board games so we all knew what to do”

Table 4: Evaluation of game features

Feature	Average (1 = very bad, 5 = very good; or 1 = strongly disagree, 5 = strongly agree)
Written instructions	Average = 4.44, SD = 0.771
The game was interesting	Average = 4.37, SD = 0.963
Realistic scenario	Average = 4.37, SD = 0.692
Game logic was apparent	Average = 4.18, SD = 0.730
Useful to work in teams	Average = 4.15, SD = 0.714
Prefer game-board version	Average = 3.98, SD = 0.754

Outside of this research project, seven players had played The Beer Game mentioned before. In The Beer Game all calculations are performed by hand on simple worksheets. This found favour:

“Doing the calculations by hand means we have to understand”
“The calculator half of the game hides details. Just give us a calculator and we can work it out”

Although the players’ reception of the game was generally positive, clear written instructions are essential to make sure best use is made of the game session time. This comment was made by a player in the very first game session:

“Wasn’t sure of what we were supposed to do”

Initially, instructions for playing the game were delivered by the researcher after the players had completed the pre-game survey and just before they started the game. For the second game session onwards, a one-page instruction sheet was emailed to each player a couple of days beforehand so they could be prepared.

The database of Simsoft game transactions showed that only three games had to be abandoned and restarted. It was observed that once teams had made the first couple of decisions, they were able to continue with too much trouble.

4.2 Working in Groups

An important component of many of the pedagogical theories behind Simsoft is the aspect of working in groups or teams, so it was important to assess how this was received by the players. A majority of players (44 out of 59) said they found it useful or very useful to work as a team

and that this reflected how things often happened in the workplace:

“It was like [the agile] stand up meeting we have every morning”
“We organised our selves into roles we felt comfortable with or that fitted our day-job: someone on the calculator, someone moving the developer pieces, someone moving the units of work”

However, one student found something new in the practice:

“I thought software development was a solitary experience but it's not really”

Others liked the opportunity to share opinions and learn from more experienced peers:

“Everyone had a chance to offer an opinion”
“I have little real-world project experience so it was good to get the advice of others and see how they approached problems”

But, as in any group activity, the game facilitator needs to be aware of cultural differences that may make some less inclined to contribute and of players who are dominating their groups:

“Generally, everyone had their say in final decision but a couple of times we were overridden”

4.3 Summary

These are the initial findings discovered through a series of Simsoft game sessions conducted with teams of post-graduate project management students, and practising software project managers and developers.

The first initial finding was that the majority of the participants found working in groups was a positive experience. The participants were a diverse group of cultures, skills, and experience and many felt they were still able to work out collaborative decisions in a constructive manner. However, as with any group activity, facilitators need to be cognizant of any individuals dominating a group or others who might need a gentle prompt to contribute more.

The second initial finding was a majority of participants preferred to play around a game board rather than a fully computerized game because this was a familiar and simple activity and less time was lost to overcoming technological problems and to making simple ergonomic arrangements such as fitting all the team around a single computer. Even so, facilitators need to prepare the participants for the game

sessions by giving them clear instructions and sufficient lead time to absorb the information.

These findings were reviewed by four participants chosen at random and all concurred without comment.

5. Conclusions

Preparing students for employment is of paramount importance for universities. Not only can this help improve employment prospects but it can also better meet employer expectations. A capstone project is designed to assist with this transition to employment. However, prior to undertaking a capstone project there are potentially significant pedagogical benefits to teaching project management using a game such as Simsoft. Importantly, the interim results presented in this paper demonstrate that even simple games can help students experience the team work, negotiation, and consensus-building skills they will need in the workforce.

References

- [1] Joint Task Force on Computing Curriculum, Software Engineering 2004: Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering, IEEE Computer Society/Association for Computing Machinery, 2004.
- [2] B.S. Bloom, B.B. Masia, D.R. Krathwohl, Taxonomy of Educational Objectives: The Classification of Educational Goals, Handbook I: Cognitive Domain ed., Longman, London, 1956.
- [3] J.P. Gee, Situated Language and Learning: A Critique of Traditional Schooling, Routledge, London, 2004.
- [4] M. Savin-Baden, C.H. Major, Foundations of Problem-Based Learning, The Society for Research into Higher Learning & Open University Press, Maidenhead, 2004.
- [5] C. Aldrich, Learning by Doing: A Comprehensive Guide to Simulations, Computer Games, and Pedagogy in e-Learning and Other Educational Experiences Pfeiffer, San Francisco, 2005.
- [6] R.F. Baumeister, C. Finkenauer, Review of General Psychology, 5 (2001) 323 – 370.
- [7] B. Suits, Ethics, 77 (1967) 209 – 213.
- [8] C.C. Abt, Serious Games, The Viking Press, New York, 1970.
- [9] M. Schrage, T. Peters, Serious Play : How the World's Best Companies Simulate to Innovate, Harvard Business School Press, 1999.
- [10] D. Michael, S. Chen, Serious Games: Games That Educate, Train, and Inform, Thomson Course Technology PTR, Boston, 2005.
- [11] J.D. Sterman, Business Dynamics: Systems Thinking and Modelling for a Complex World, Irwin McGraw-Hill, New York, 2000.
- [12] J.S. Goodwin, S.G. Franklin, Journal of Management Development, 13 (1994) 7 – 15.
- [13] E. Mosekilde, E.R. Larsen, System Dynamics Review, 4 (1988) 131 - 147.
- [14] J.D. Sterman, Management Science, 35 (1989) 321 – 339.
- [15] C.W. Caulfield, S.P. Maj, in: M. Iskander (Ed.) Innovative Techniques in Instruction Technology, E-learning, E-assessment, and Education, Springer, 2007, pp. 86 – 91.
- [16] J. Wolfe, Decision Sciences, 9 (1978) 143 – 155.
- [17] A.P. Raia, The Journal of Business, 39 (1966) 339 – 352.
- [18] K.E.F. Watt, Simulation, 28 (1977) 1 – 3.
- [19] R.J. Butler, T.F. Pray, D.R. Strang, Decision Sciences, 10 (1979) 480 – 486.



Craig Caulfield is a senior software engineer for a technology consulting company and PhD candidate at Edith Cowan University. His research areas include problem-based learning and the application of serious games to software engineering education and project planning.



Dr. David Veal is a Senior Lecturer at Edith Cowan University. He is the manager of Cisco Network Academy Program at Edith Cowan University and be a unit coordinator of all Cisco network technology units. His research interests are in Graphical User Interface for the visually handicapped and also computer network modeling.



A/Prof S. P. Maj has been highly successful in linking applied research with curriculum development. In 2000 he was nominated ECU University Research Leader of the Year award He was awarded an ECU Vice-Chancellor's Excellence in Teaching Award in 2002, and again in 2009. He received a National Carrick Citation in 2006 for *"the development of world class curriculum and the design and implementation of associated world-class network teaching laboratories"*.

A Systematic Survey of Games Used for Software Engineering Education

Craig Caulfield (Corresponding author)

School of Computer Science and Security Science, Edith Cowan University
2 Bradford Street, Mount Lawley, Western Australia, 6050, Australia
Tel: 61-8-9370-6295 E-mail: ccaulfie@our.ecu.edu.au

Jianhong (Cecilia) Xia

Department of Spatial Sciences, Curtin University
Kent Street, Bentley, Western Australia, 6102, Australia
Tel: 61-8-9266-7563 E-mail: c.xia@curtin.edu.au

David Veal

School of Computer Science and Security Science, Edith Cowan University
2 Bradford Street, Mount Lawley, Western Australia, 6050, Australia
Tel: 61-8-9370-6295 E-mail: d.veal@ecu.edu.au

S Paul Maj

School of Computer Science and Security Science, Edith Cowan University
2 Bradford Street, Mount Lawley, Western Australia, 6050, Australia
Tel: 61-8-9370-6277 E-mail: p.maj@ecu.edu.au

Received: September 7, 2011 Accepted: October 9, 2011 Published: December 1, 2011

doi:10.5539/mas.v5n6p00

URL: <http://dx.doi.org/10.5539/mas.v5n6p00>

Abstract

Simsoft is a serious game—one that trains or educates—at the centre of a research project designed to see if and how games can contribute to better software engineering management education by helping software engineers and project managers explore some of the dynamic complexities of the field in a safe and inexpensive environment. A necessary precursor for this project was to establish what games already existed in the field and how effective they had been. To this end a systematic review of the literature was conducted using a collection of online science, engineering, education, and business databases looking for games or simulations used for educational or training purposes in software engineering or software project management across any of the SWEBOK knowledge areas. The initial search returned 243 results, which was filtered to 36 papers by applying some simple quality and relevance inclusion/exclusion criteria. These remaining papers were then analysed in more depth to see if and how they promoted education in the field of software engineering management. The results showed that games were mainly used in the SWEBOK knowledge areas of software engineering management and development processes, and most game activity was in Europe and the Americas. The results also showed that most games in the field have learning objectives pitched at the first rung of Bloom's taxonomy (knowledge), most studies followed a non-experimental design, and many had very small sample sizes. This suggests that more rigorous research is needed into the efficacy of games in teaching software engineering management, but enough evidence exists to say that educators could include serious games in their courses as a useful and interesting supplement to other teaching methods.

Keywords: Software engineering, Project management education, Serious games

1. Introduction

1.1 Defining Games

To play a game, “is to engage in activity directed towards bringing about a specific state of affairs, using only means permitted by specific rules, where the means permitted by the rules are more limited in scope than they would be in the absence of the rules and where the sole reason for accepting such limitation is to make possible such activity” (Suits, 1967, p. 156).

A game is different from a model or simulation. To start, a model is “a miniature representation of a complex reality. A model reflects certain selected characteristics of the system it stands for. A model is useful to the extent that it portrays accurately those characteristics that happen to be of interest at the moment” (DeMarco, 1982, p. 14). Meanwhile, a simulation is a special kind of model that exhibits processes in some way like the system it represents, and that shows how these processes change from state A to state B, between two points in time (J. G. Miller, 1978, p. 83).

Games naturally come in many forms. In a seminal work in the field, *Man, Play and Games*, Caillois (1961) proposed a classification that depends on whether the role of competition (*agôn*), chance (*alea*), simulation (*mimicry*), or vertigo (*ilinx*) is dominant. *Agôn* are those games “that would seem to be competitive... like a combat in which equality of chances is artificially created in order that the adversaries should confront each other under ideal conditions” (Caillois, 1961, p. 14). Football, billiards, or chess fall into this category. *Alea* are games of chance such as roulette or a lottery; games of mimicry involve the players becoming other characters, such as cowboys and Indians; while *ilinx* are “those which are based in the pursuit of vertigo and which consists of an attempt to momentarily destroy the stability of perception and inflict a kind of voluptuous panic upon an otherwise lucid mind” (Caillois, 1961, p. 23).

The games that this research project deals with are a subset of Caillois’s *agôn* classification and they use an adjective—serious—to show they want for more than simple amusement and that they are designed to educate, train, or inform their players (Abt, 1970; Michael & Chen, 2005; Schrage & Peters, 1999).

1.2 The Value of Games

Games have been used to train and educate players for many years in many different fields (see for example, Gee, 2007b; Michael & Chen, 2005; Perla, 1990; Prensky, 2007) and are based on learning and development theories such as problem-based learning (Savin-Baden & Major, 2004), experiential education (Dewey, 1938/1963; Kolb, 1984; Papert, 1980), and decision science (Raser, 1969, pp. 46-55). Yet, to a common extent, games have been found to be more expensive and administratively demanding to develop and use than some other forms of instruction or research (Abt, 1970, pp. 110-111; Babb, Leslie, & Van Syke, 1966, p. 471; Cohen & Rhenman, 1961, p. 151; Petranek, 1994). Still, there are some offsetting advantages.

For example, it has been noted that the human capacity to understand the implications of our mental models and to accurately trace through even a small number of causal relationships is fairly limited (G. A. Miller, 1956; Simon, 1957). Yet, a game is a visible and physical representation of a problem space; a captured mental model. As such, they are places to trial new ideas and to experiment with established theories (Feldman, 1995; McKenney, 1962); to replay these theories as many times as needed; places where time and space can be contracted or expanded (Raser, 1969); places where it is acceptable just to try different things and where more might be learned from failure than success (Booker, 1994).

Even so, there are some dangers to be heeded when using games. Games are just... games, and as such are just one representation of how the world works. Therefore, “it is potentially dangerous to have players leave the gaming environment with the belief that the strategies that were effectively employed in playing the game are directly transferable to the real world” (Watson & Blackstone, 1989, p. 493). Participants should ideally be provided with more information than just the game to help them wisely discriminate between what may or may not work outside the game itself (Andlinger, 1958, pp. 152-158).

It was with these pros and cons aforethought that a game—Simsoft (Caulfield, Veal, & Maj, 2011b)—was developed to see what value games might bring to the education of software engineers and project managers.

1.3 Simsoft

Simsoft comes in two pieces. There is an A0-sized printed game board around which the players gather to discuss the current state of their project and to consider their next move. The board shows the flow of the game while plastic counters are used to represent the staff of the project. Poker chips represent the team’s budget, with which they can purchase more staff, and from which certain game events may draw or reimburse amounts depending on decisions made during the course of the game. There is also a simple Java-based dashboard (Caulfield, Veal, & Maj, 2011a), through which the players can see the current and historical state of the project

in a series of reports and messages; and they can adjust the project's settings. The engine behind Simsoft is a system dynamics model which embodies a small set of fundamental causal relationships of simple software development projects.

In Simsoft game sessions, teams of students, and practicing project managers and software engineers managed a hypothetical software development project with the aim of completing the project on time and within budget (with poker chips left over). Based on the starting scenario of the game, information provided during the game, and their own real-world experience, the players made decisions about how to proceed— whether to hire more staff or reduce the number, what hours should be worked, and so on. After each decision set had been entered, the game was run for another next time period, (a week, a month, or a quarter). The game was now in a new state which the players had to interpret from the game board and decide how to proceed.

A necessary precursor for this project was find out what games already existed in the field of software engineering education, how effective they had been, and how Simsoft might be able to contribute new knowledge. To this end a systematic review of the literature was conducted using a collection of online science, engineering, education, and business databases looking for games or simulations used for educational or training purposes in software engineering or software project management across any of the Software Engineering Body of Knowledge (SWEBOK (Bourque, Dupuis, Abran, Moore, & Tripp, 1999)) knowledge areas.

2. Survey Methods

For this survey, we followed an established procedure for conducting systematic reviews in the field of software engineering (Kitchenham, 2004), which has been used to survey the game field before (Gresse von Wangenheim & Shull, 2009). Given the upward trend in the use of games for software engineering education revealed in that previous survey, it was timely to update and expand the search.

2.1 Data Sources and Search Strategy

To perform this review we used the IEEE Xplore Digital Library, the ACM Digital Library, ScienceDirect, Sage Journals Online, ProQuest, the ISI Web of Knowledge, and the Wiley Online Library. The following pseudo-code search string was adapted for the specific query languages of each library:

where abstract OR title OR keywords contain

((game OR simulation) AND (learning OR teaching OR education OR training))

AND

(software engineering OR software project OR

software process OR software design OR

software testing OR software configuration management OR

software quality OR software management OR

software maintenance OR software construction

OR software requirements OR software engineering tools and methods))

AND

(date >= 1990)

That is, we looked for games or simulations (computer and non-computer based) used for educational or training purposes in software engineering or software project management across any of the SWEBOK knowledge areas. (Despite the distinction made between game and simulation in the introduction, the terms are often used interchangeably in the literature (Maier & Grossler, 2000), therefore *simulation* has been used as one of the search parameters).

2.2 Inclusion and Exclusion Criteria

We limited the results to English-language papers published from 1990 to the present in peer-reviewed journals and conference proceedings. We excluded position papers, papers in which no data was reported (unless they were preliminary papers for completed studies), and those in which the game or simulation was not used to train or educate the players at a tertiary level.

2.3 Study Identification and Selection

The initial database searches returned a total of 243 papers. The titles and abstracts were analysed according to the inclusion and exclusion criteria, and any off-topic papers were discarded. This left 36 papers, which were

grouped according to the study they described.

2.3 Data Extraction

Each paper passing the selection process was read in depth and the following data was extracted:

- References to the papers describing the study.
- A brief description of the game and how it was played.
- The experimental design used by the study, which could be either true experimental (random assignment and comparison with a control group), quasi-experimental (comparison with a control group only), or non-experimental.
- The number and type of the players.
- The type of research tool used to collect the data, for example questionnaires, observation, pre- and post-test surveys.
- The primary SWEBOK knowledge area on which the game is focussed. The SWEBOK attempts to characterise and bound the software engineering body of knowledge; the ten knowledge areas are the major topical divisions within the field.
- The expected learning outcomes classified according to Bloom's (1956) cognitive domain taxonomy. The cognitive domain defines six incremental levels of learning objectives that educators may have for their students: *knowledge*: remember previously-learned materials by recalling specific facts, terminology, theories and answers; *comprehension*: demonstrate an understanding of information by being able to compare, contrast, organize, interpret, describe, and extrapolate; *application*: use previously-learned material in new situations; *analysis*: decompose previously-learned material into parts in order find patterns and to make inferences and generalizations; *synthesis*: use existing ideas in different ways to create new ideas or to propose alternative solutions; *evaluation*: judge the validity of ideas or information with a certain context.
- The principal findings of the study.
- The country in which the game sessions were conducted.

Table 1 shows the full data extract of 36 papers describing 26 studies.

3. Survey Results

Figure 1 shows that the preferred medium for games in the field is computer-based (22 out of 26) rather than other types such as board and card games. This way the games are easier to distribute and administer across a large number of players who may be in remote locations. Figure 1 also shows that most of the studies were non-experimental (16 out of 26) meaning they didn't use control groups nor randomly assign participants to different groups.

The survey results show that games have been used in a variety of ways to teach different aspects of software engineering and software project management. Figure 2 shows the distribution of games across the world based on the SWEBOK knowledge area they were designed to address. Most games (21 out of 26) focused broadly on software engineering management or the development process and most activity (21 out of 26) occurred in Europe and the Americas.

Figure 1 shows that overwhelmingly, the learning objectives of the studies pitched at the first rung of Bloom's taxonomy, knowledge. In general, those studies that assessed the degree of learning by the participants found that the participants were sometimes learning new concepts, but they were mainly reinforcing known theories. All the research projects, whether explicitly or implicitly stated, found that games alone were not sufficient pedagogical devices to teach software engineering or project management concepts and would have to be supplemented by other means. Only Navarro (2009) and Hainey et al. (2010) evaluated the effectiveness of games for players of different skills and backgrounds and each found that games were suitable for a wide variety of participants.

It should be noted, however, that apart from Navarro's and Drappa and Ludewig's body of work, many of the research projects in Table 1 had very small sample sizes and few others were developed or repeated beyond that described in the initial papers.

4. Simsoft Compared to Other Games in the Field

Recalling the discussion of model, simulation, and game given at the beginning of this paper: a *model* is a convenient representation (in words, numbers, or other symbols) of some real-world socio-economic or socio-technical system; a *simulation* is dynamic, operational model through which changes over time are revealed; and

a *game* is a simulation that is purposefully run, wholly or partly determined by players' decisions, within some predetermined circumstances. It can be said that software development has been *modelled* (Belady & Lehman, 1976; Boehm, 1981; Boehm et al., 2000; McCabe, 1976; H. Remus & Zilles, 1979) and *simulated* (Abdel-Hamid & Madnick, 1991; Collofello, 2000; Hansen, 1996; Madachy, 2008; Raffo, 1996; Tvedt, 1996; Variale, Rosetta, Steffen, Rubin, & Yourdon, 1994) many times. But, these are not the software engineering perspectives of interest here because:

- They focus primarily on predicting rather than educating. For example, Boehm's COCOMO model (2000) is designed to calculate the cost and effort of a software project based on historical data and what is currently known about the project at hand. COCOMO is used to validate an estimate, not necessarily find out why it is this number.
- They are not interactive or designed for group participation. For example, perhaps the most well-known simulation (Abdel-Hamid & Madnick, 1991) contains over 300 underlying variables, doesn't have a way to interact with the model except through direct manipulation of these variables at a source code level, and still does not describe the development process in detail (Martin, 2002, pp. 32-37).

Given their focus, it is not surprising that these models and simulations fail most, if not all, of Gee's principles of interactive game design (Caulfield, Veal, & Maj, 2011c; Gee, 2007a, 2007b). In contrast, the games described in Table 1 more closely align with Gee's principles. Still, there are differences between these games and Simsoft.

SimSE, the game developed by Navarro (2009) and her colleagues at the University of California, Irvine over a number of years, is perhaps the most advanced game in the field and the only one in Table 1 that has been developed much beyond its initial implementation. SimSE supports a number of different development methodologies (such as rapid prototyping, inspection, and the Rational Unified Process), provides users with a performance report after they complete the game, and has also been tested and verified in a range of controlled classroom settings. Players manage their SimSE project through a rich graphical user interface that shows their team at work along with various management reports and dials. In contrast to Simsoft, SimSE is a single-user game so without players clustering around a single screen, there's little opportunity to discuss and debate project decisions and come to a consensus. SimSE is also heavily focussed on the process of software development—the *how* of software development—whereas Simsoft is also concerned with the *who*.

Like Simsoft, a number of the games in the field have eschewed computers, either completely or partly, in favour of playing cards, boards, and sometimes dice. For example, in Zapata's (2010) game, teams throw a dice, that determines which of a collection of technical questions the team must answer. From here, the team gets a chance to estimate the size of a project component and score points. This slightly convoluted game show format relies more on chance than skill and means that most players are dormant and passive while other teams are having their turn. Chance also plays a role in games like Problems and Programmers (Baker, Oh Navarro, & van der Hoek, 2005)—players draw cards from a shuffled deck—and PlayScrum (Fernandes & Sousa, 2010)—a roll of the dice determines what resources the player can accumulate and what problems may be encountered. Unlike Simsoft, these games are competitive rather than co-operative.

Some of the games in Table 1 offer only a very high level of interactivity meaning players can perform just broad project functions and hence only see general project dynamics. In SimVBSE (Jain & Boehm, 2006), SimjavaSP (Shaw & Dermoudy, 2005), MO-SEProcess (Zhu, Wang, & Tan, 2007), Hainey's game (2010), and OSS (Sharp & Hall, 2000) players make their avatar visit certain rooms or characters to ask questions or collect information. In Hainey's game the result of this office tour is a requirements document that is then passed to the project manager avatar for assessment. The tour may have to be repeated if all the requirements haven't been identified. A game interface makes this engaging for a while, but how it relates to real-world software project management is dubious. Providing the same information in a short project description, such as the one that comes with Simsoft, means the player can begin exploring the problem domain sooner. And, with less effort required to create the office environment, more could be devoted to the interesting detail of the project's dynamics.

SESAM (Drappa & Ludewig, 1999; Drappa & Ludewig, 2000) could almost be called a model or simulation rather than a game because a user runs it by typing commands in a complex modelling language and the system responds in kind. In exchange for this complexity, SESAM allows its users to define a wide variety of development methodologies as well as hire and fire staff, assign tasks, and ask developers about their progress. But, without an effective visual interface, playing SESAM is like programming an old VCR: there isn't enough feedback to know what is happening (Norman, 1988, pp. 51-53). It is perhaps not surprising that SESAM has not been developed far beyond that described in the original papers. In contrast, Simsoft's state of play is always visible on the game board.

One feature common to all the projects in Table 1 is the research population they use: the participants are either

undergraduate or post-graduate university, and in one case high school, students. In broader research circles, there is some debate (Camerer & Johnson, 1991; Garb, 1989; Remus, 1986) about whether students make viable candidates for research involving management decisions because they may lack the experience and knowledge to make their responses transferable to the workplace. Simsoft side-steps this still inconclusive debate because its research population is a mixture of university students and project managers and software developers of varying lengths of experience.

In summary, there are four main differences between the approach taken in this research project and others in the area:

- Simsoft is equally, if not more, concerned with *who* does the work in a software development as it is with process of *how* the work is done. This echoes the cover of Boehm's (1981) *Software Engineering Economics* which shows personnel is where the greatest productivity gains are possible.
- Simsoft is largely a board game (with a small calculator component) in contrast to other games that use a graphical user interface of varying levels of richness. Often the user interface is simply a conceit of the game for performing housekeeping functions and lends little to the real purpose. Other games that use playing cards or games boards contain an element of chance rather than skill.
- Simsoft is cast at a level of detail at which the players can see the movement of individual pieces of work and individuals themselves. Games cast at higher levels, such as OSS, mask some fundamental project dynamics.
- The research sample for this project is a mixture of students and experienced professionals rather than wholly students.

5. Conclusions

This systematic survey of games used in software engineering management education has shown that, as a pedagogical device, they are becoming more common, particularly in Europe and the Americas, and students in general enjoyed playing them and felt they got some value from the experience. However, few of the games were developed beyond their initial implementations suggesting their pedagogical value was not demonstrated sufficiently.

From these findings, there are some implications for researchers, educators, and game developers:

- More rigorous research is needed into the efficacy of games in teaching software engineering management. Most of the games in Table 1 didn't follow a true experimental design and many had very small sample sizes, meaning the findings should be viewed with some caution.
- Even so, enough evidence exists to suggest that educators should consider using games as part of their courses in software engineering, but as an interesting supplement to other teaching materials and preferably later in the course when the students have had time to gain the knowledge needed to make sense of what the game is trying to teach.
- In many of the games in Table 1, rich graphics and avatars contributed little to meeting the learning objectives of the game and sometimes distracted or frustrated the players. Making the games simpler would shorten the time it takes to create the games and also allow the players to focus more on the content.

These findings have influenced the design and implementation of Simsoft, the serious game behind this research project. For example, Simsoft is a simple, collaborative board game, which has so far been played by combined teams of students and experienced software developers and project managers. Further games sessions are under way to test the efficacy of the current implementation.

References

- Abdel-Hamid, T. K., & Madnick, S. E. (1991). *Software Project Dynamics: An Integrated Approach*. Englewood Cliffs: Prentice-Hall.
- Abt, C. C. (1970). *Serious Games*. New York: The Viking Press.
- Andlinger, G. R. (1958). Looking Around: What Can Business Games Do?. *Harvard Business Review*. 36(4), 147–160.
- Babb, E. M., Leslie, M. A., & Van Syke, M. D. (1966). The Potential of Business Gaming Methods in Research. *The Journal of Business*. 39(4), 465–472. <http://dx.doi.org/10.1086/294887>
- Baker, A., Navarro, E. O., & van der, H., Andre. (2003). Problems and Programmers: An Educational Software Engineering Card Game. *Proceedings of The 25th International Conference on Software Engineering (ICSE'03)*. 3-10 May, 2003. Portland: Oregon. <http://doi.ieeecomputersociety.org/10.1109/ICSE.2003.1201245>

- Baker, A., Oh Navarro, E., & van der Hoek, A. (2005). An Experimental Card Game for Teaching Software Engineering Processes. *The Journal of Systems and Software*. 75(1-2). <http://dx.doi.org/10.1016/j.jss.2004.02.033>
- Barros, M. d. O., Dantas, A. R., Veronese, G. O., & Werner, C. M. L. (2006). Model-Driven Game Development: Experience and Model Enhancements in Software Project Management Education. *Software Process: Improvement and Practice*. 11(4), 411-421. <http://dx.doi.org/10.1002/spip.279>
- Belady, L. A., & Lehman, M. M. (1976). A Model of Large Program Development. *IBM Systems Journal*. 15(3), 225-252. <http://dx.doi.org/10.1147/sj.153.0225>
- Bloom, B. S., Masia, B. B., & Krathwohl, D. R. (1956). *Taxonomy of Educational Objectives: The Classification of Educational Goals* (Handbook I: Cognitive Domain ed.). London: Longman.
- Boehm, B. W. (1981). *Software Engineering Economics*. Sydney: Prentice-Hall.
- Boehm, B. W., Abts, C., Brown, A. W., Chulani, S., Clark, B. K., Horowitz, E., et al. (2000). *Software Cost Estimation with Cocomo II*. Upper Saddle River: Prentice Hall.
- Booker, E. (1994, 4 July). Have You Driven a Simulated Ford Lately? *Computerworld*. 28, 76.
- Bourque, P., Dupuis, R., Abran, A., Moore, J. W., & Tripp, L. (1999). The Guide to the Software Engineering Body of Knowledge. *IEEE Software*. 16(6), 35-44. <http://dx.doi.org/10.1109/52.805471>
- Caillois, R. (1961). *Man, Play and Games* (M. Barash, Trans.). New York: Free Press of Glencoe.
- Camerer, C. F., & Johnson, E. J. (1991). The Process-Performance Paradox in Expert Judgment. In K. A. Ericsson & J. Smith (Eds.). *Toward a General Theory of Expertise: Prospects and Limits*. (pp. 195-217). Cambridge: Cambridge University Press.
- Caulfield, C. W., Veal, D., & Maj, S. P. (2011c). Teaching Software Engineering Project Management—A Novel Approach for Software Engineering Programs. *Modern Applied Science*. 5(5). in press.
- Caulfield, C. W., Veal, D., & Maj, S. P. (2011a). Implementing System Dynamics Models in Java. *International Journal of Computer Science and Network Security*. 11(7), 43-49.
- Caulfield, C. W., Veal, D., & Maj, S. P. (2011b). Teaching Software Engineering Management – Issues and Perspectives. *IJCSNS International Journal of Computer Science and Network Security*. 11(7), 50-54.
- Cohen, K. J., & Rhenman, E. (1961). The Role of Management Games in Education and Research. *Management Science*. 7, 131-166. <http://dx.doi.org/10.1287/mnsc.7.2.131>
- Collofello, J. (2000). University/Industry Collaboration in Developing a Simulation Based Software Project Management Training Course. *Proceedings of Proceedings of the Thirteenth Conference on Software Engineering Education & Training*. Austin: Texas.
- Connolly, T. M., Stansfield, M., & Hainey, T. (2007). An Application of Games-Based Learning within Software Engineering. *British Journal of Educational Technology*. 38(3), 416-428. <http://dx.doi.org/10.1111/j.1467-8535.2007.00706.x>
- Dantas, A. R., Barros, M. d. O., & Werner, C. M. L. (2004). A Simulation-Based Game for Project Management Experiential Learning. *Proceedings of The Sixteenth International Conference on Software Engineering & Knowledge Engineering*. Banff: Alberta, Canada.
- DeMarco, T. (1982). *Controlling Software Projects*. New York: Yourdon Press.
- Dewey, J. (1938/1963). *Experience and Education*. New York: Collier Books.
- Drappa, A., & Ludewig, J. (1999). Quantitative Modeling for the Interactive Simulation of Software Project. *The Journal of Systems and Software*. 46(15 April), 113-122. [http://dx.doi.org/10.1016/S0164-1212\(99\)00005-9](http://dx.doi.org/10.1016/S0164-1212(99)00005-9)
- Drappa, A., & Ludewig, J. (2000). Simulation in Software Engineering Training. *Proceedings of The 22nd International Conference on Software Engineering*. Limerick, Ireland. <http://dx.doi.org/10.1145/337180.337203>
- Feldman, H. D. (1995). Computer-Based Simulation Games: A Viable Educational Technique for Entrepreneurship Classes?. *Simulation & Gaming*. 26(3), 346-360. <http://dx.doi.org/10.1177/1046878195263006>
- Fernandes, J. M., & Sousa, S. M. (2010). PlayScrum - A Card Game to Learn the Scrum Agile Method. *Proceedings of The 2010 Second International Conference on Games and Virtual Worlds for Serious Applications*.

- Garb, H. N. (1989). Clinical Judgment, Clinical Training, and Professional Experience. *Psychological Bulletin*. 105(3), 387–396.
- Gee, J. P. (2007a). *Good Video Games and Good Learning: Collected Essays on Video Games, Learning and Literacy*. New York: Peter Lang Publishing.
- Gee, J. P. (2007b). *What Video Games Have to Teach Us About Learning and Literacy*. New York: Palgrave MacMillan.
- Gresse von Wangenheim, C., & Shull, F. (2009). To Game or Not to Game?. *IEEE Software*. 26(2), 92 – 94. <http://doi.ieeecomputersociety.org/10.1109/MS.2009.54>
- Gresse von Wangenheim, C., Thiry, M., & Kochanski, D. (2009). Empirical Evaluation of an Educational Game on Software Measurement. *Empirical Software Engineering*. 14(4), 418-452. <http://dx.doi.org/10.1007/s10664-008-9092-6>
- Hainey, T., Connelly, T. J., Stansfield, M., & Boyle, E. A. (2010). Evaluation of a Game to Teach Requirements Collection and Analysis in Software Engineering at Tertiary Education Level. *Computers & Education*. 56(1), 21–35. <http://dx.doi.org/10.1016/j.compedu.2010.09.008>
- Hansen, G. A. (1996). Simulating the Software Development Process. *IEEE Computer*. 29(1), 73–77. <http://doi.ieeecomputersociety.org/10.1109/2.481468>
- Jain, A., & Boehm, B. (2006). SimVBSE: Developing a Game for Value-Based Software Engineering. *Proceedings of Proceedings of the 19th Conference on Software Engineering Education & Training*. <http://dx.doi.org/10.1109/cseet.2006.31>
- Kitchenham, B. A. (2004). *Procedures for Performing Systematic Reviews*. Keele University, Staffordshire.
- Knauss, E., Schneider, K., & Stapel, K. (2008). A Game for Taking Requirements Engineering More Seriously. *Proceedings of The Third International Workshop on Multimedia and Enjoyable Requirements Engineering - Beyond Mere Descriptions and with More Fun and Games*. 9 September, 2008. Barcelona, Spain. <http://doi.ieeecomputersociety.org/10.1109/MERE.2008.1>
- Kolb, D. A. (1984). *Experiential Learning: Experience as the Source of Learning and Development*. Englewood Cliffs: Prentice-Hall.
- Madachy, R. J. (2008). *Software Process Dynamics*. Hoboken: John Wiley & Sons.
- Maier, F. H., & Grossler, A. (2000). What Are We Talking About? — A Taxonomy of Computer Simulations to Support Learning. *System Dynamics Review*. 16(2), 135–148. [http://dx.doi.org/10.1002/1099-1727\(200022\)16:2<135::AID-SDR193>3.0.CO;2-P](http://dx.doi.org/10.1002/1099-1727(200022)16:2<135::AID-SDR193>3.0.CO;2-P)
- Mandl-Striegnitz, P. (2001). *How to Successfully Use Software Project Simulation for Educating Software Project Managers*. Proceedings of The Frontiers in Education Conference.
- Martin, R. C. (2002). *Agile Software Development: Principles, Patterns, and Practices*. Upper Saddle River: Prentice-Hall.
- McCabe, T. J. (1976). A Software Complexity Measure. *IEEE Transactions on Software Engineering*. 2(4), 308–320. <http://doi.ieeecomputersociety.org/10.1109/TSE.1976.233837>
- McKenney, J. L. (1962). An Evaluation of a Business Game in an MBA Curriculum. *The Journal of Business*. 35(3), 278–286.
- Michael, D., & Chen, S. (2005). *Serious Games: Games That Educate, Train, and Inform*. Boston: Thomson Course Technology PTR.
- Miller, G. A. (1956). The Magical Number Seven, Plus or Minus Two: Some Limits on Our Capacity for Processing Information. *Psychological Review*. 63(2), 81–97.
- Miller, J. G. (1978). *Living Systems*. New York: McGraw-Hill Book Company.
- Navarro, E. O. (2006). *SimSE: A Software Engineering Simulation Environment for Software Process Education*. Unpublished Thesis. University of California, Irvine.
- Navarro, E. O., & van der Hoek, A. (2005). Design and Evaluation of an Educational Software Process Simulation Environment and Associated Model. *Proceedings of The Eighteenth Conference on Software Engineering Education and Training*. Ottawa, Canada.

- Navarro, E. O., & van der Hoek, A. (2007). Comprehensive Evaluation of an Educational Software Engineering Simulation Environment. *Proceedings of The Twentieth Conference on Software Engineering Education and Training*. July 2007.
- Navarro, E. O., & van der Hoek, A. (2008). On the Role of Learning Theories in Furthering Software Engineering Education. In H. J. C. Ellis, S. A. Demurjian & J. F. Naveda (Eds.). *Software Engineering: Effective Teaching and Learning Approaches and Practices* IGI Global. IGI Global.
- Navarro, E. O., & van der Hoek, A. (2009). Multi-Site Evaluation of SimSE. *Proceedings of The 40th ACM Technical Symposium on Computer Science Education*. March 3–7. Chattanooga, Tennessee.
- Navarro, E. O., Baker, A., & van der Hoek, A. (2004). Teaching Software Engineering Using Simulation Games. *Proceedings of The 2004 International Conference on Simulation in Education*. January 2003 San Diego, California
- Norman, D. A. (1988). *The Psychology of Everyday Things*. New York: Basic Books.
- Oh, E., & van der Hoek, A. (2002). Towards game-Based Simulation as a Method of Teaching Software Engineering. *Proceedings of The Frontiers in Education. 2002. FIE 2002*. 32nd Annual, 6-9 Nov. 2002. <http://dx.doi.org/10.1109/FIE.2002.1158674>
- Papert, S. (1980). *Mindstorms*. Brighton. Sussex: The Harvester Press.
- Perla, P. P. (1990). *The Art of Wargaming: A Guide for Professionals and Hobbyists*. Annapolis, Maryland: Naval Institute Press.
- Petranek, C. F. (1994). A Maturation in Experiential Learning: Principles of Simulation and Gaming. *Simulation & Gaming*. 25(4), 513–522, <http://dx.doi.org/10.1177/1046878194254008>.
- Pfahl, D., Koval, N., & Ruhe, G. (2001). An Experiment for Evaluating the Effectiveness of Using a System Dynamics Simulation Model in Software Project Management Education. *Proceedings of The Software Metrics Symposium, 2001. METRICS 2001. Proceedings. Seventh International*. <http://dx.doi.org/10.1109/METRIC.2001.915519>
- Pfahl, D., Laitenberger, O., Dorsch, J., & Ruhe, G. (2003). An Externally Replicated Experiment for Evaluating the Learning Effectiveness of Using Simulations in Software Project Management Education. *Empirical Software Engineering*. 8(4), 367–395. <http://dx.doi.org/10.1023/a:1025320418915>
- Pfahl, D., Laitenberger, O., Ruhe, G., Dorsch, J., & Krivobokova, T. (2004). Evaluating the Learning Effectiveness of Using Simulations in Software Project Management Education: Results from a Twice Replicated Experiment. *Information and Software Technology*. 46(2), 127-147. <http://www.sciencedirect.com/science/article/pii/S0950584903001150>
- Prensky, M. (2007). *Digital Game-Based Learning*. St. Paul, Minnesota: Paragon House Publishers.
- Raffo, D. M. (1996). *Modeling Software Processes Quantitatively and Assessing the Impact of Potential Process Changes on Process Performance (TQM)*. Unpublished Thesis, Carnegie-Mellon University, Pittsburgh, Pennsylvania.
- Raser, J. R. (1969). *Simulation and Society: An Exploration of Scientific Gaming*. Boston: Allyn and Bacon Inc.
- Remus, H., & Zilles, S. (1979). Prediction and Management of Program Quality. *Proceedings of The 4th International Conference on Software Engineering*. Munich, Germany.
- Remus, W. (1986). Graduate Students as Surrogates for Managers in Experiments on Business Decision Making. *Journal of Business Research*. 14(1), 19–25.
- Rodriguez, D., Sicilia, M. A., Cuadrado-Gallego, J. J., & Pfahl, D. (2006). e-Learning in Project Management Using Simulation Models: A Case Study Based on the Replication of an Experiment. *Education, IEEE Transactions on*. 49(4), 451-463. <http://dx.doi.org/10.1109/TE.2006.882367>
- Rusu, A., Russell, R., Robinson, J., & Rusu, A. (2010). Learning Software Engineering Basic Concepts Using a Five-Phase Game. *Proceedings of The 40th ASEE/IEEE Frontiers in Education Conference*. October 27-30, 2010 Washington, DC. <http://dx.doi.org/10.1109/FIE.2010.5673327>
- Savin-Baden, M., & Major, C. H. (2004). *Foundations of Problem-Based Learning*. Maidenhead: The Society for Research into Higher Learning & Open University Press.
- Schrage, M., & Peters, T. (1999). *Serious Play: How the Worlds Best Companies Simulate to Innovate*. Harvard Business School Press.

- Sharp, H., & Hall, P. (2000). An Interactive Multimedia Software House Simulation for Postgraduate Software Engineers. *Proceedings of The 22nd International conference on Software engineering* Limerick, Ireland. <http://dx.doi.org/10.1145/337180.337528>
- Shaw, K., & Dermoudy, J. (2005). Engendering an Empathy for Software Engineering. *Proceedings of The 7th Australasian Conference on Computing Education*. Newcastle, New South Wales, Australia.
- Simon, H. A. (1957). *Models of Man Social and Rational: Mathematical Essays on Rational Human Behavior in a Social Setting*. New York: John Wiley & Sons.
- Suits, B. (1967). What is a Game?. *Philosophy of Science*. 34(2), 148–156.
- Taran, G. (2007). Using Games in Software Engineering Education to Teach Risk Management. *Proceedings of The Software Engineering Education & Training*. 3-5 July 2007. <http://dx.doi.org/10.1109/CSEET.2007.54>.
- Tvedt, J. D. (1996). *An Extensible Model for Evaluating the Impact of Process Improvements on Software Development Cycle Time*. Unpublished Unpublished Ph.D. dissertation, Arizona State University, Phoenix, Arizona.
- Variante, T., Rosetta, B., Steffen, M., Rubin, H., & Yourdon, E. (1994). Modeling the Maintenance Process. *American Programmer*. 7(3), 29–37
- Wang, A. I., Fsdahl, T., & Morch-Storstein, O. K. (2008). An Evaluation of a Mobile Game Concept for Lectures. *Proceedings of The 21st Conference on Software Engineering Education and Training*. 14-17 April 2008. <http://dx.doi.org/10.1109/CSEET.2008.15>
- Wang, T., & Zhu, Q. (2009). A Software Engineering Education Game in a 3-D Online Virtual Environment. *Proceedings of The First International Workshop on Education Technology and Computer Science*. 7-8 March, 2009 Wuhan, Hubei, China. <http://doi.ieeecomputersociety.org/10.1109/ETCS.2009.418>
- Watson, H. J., & Blackstone, J. H. (1989). *Computer Simulation* (2nd edition ed.). New York: John Wiley & Sons.
- Ye, E., Chang, L., & Polack-Wahl, J. A. (2007). Enhancing Software Engineering Education Using Teaching Aids in 3-D Online Virtual Worlds. *Proceedings of The Frontiers In Education Conference - Global Engineering: Knowledge Without Borders, Opportunities Without Passports*. 10-13 Oct. 2007. <http://dx.doi.org/10.1109/FIE.2007.4417884>
- Zapata, C. M. (2010). A Classroom Game for Teaching Management of Software Companies. *Dyna*. 77(163), 290–299.
- Zapata, C. M., & Awad-Aubad, G. (2007). Requirements Game: Teaching Software Project Management. *CLEI Electronic Journal*. 10(1).
- Zhu, Q., Wang, T., & Tan, S. (2007). Adapting Game Technology to Support Software Engineering Process Teaching: From SimSE to MO-SEProcess. *Proceedings of The Third International Conference on Natural Computation*. <http://dx.doi.org/10.1109/ICNC.2007.159>

Table 1. Full data extract of games used in software engineering education

ID	Study	Description	Experimental Design	Sample Size (if known) and Population	Data Collection Tool	SWEBOK Knowledge Area	Bloom Learning Outcome	Observed Learning Outcomes
GS-01	University/Industry Collaboration in Developing a Simulation Based Software Project Management Training Course. (Collofello, 2000).	A single-player game, based on a system dynamics model with an iThink user interface that models a software project. Players attempt different management exercises (risk management, life cycle model comparison, critical path scheduling, etc.) that follow the lecture material.	Non-experimental	16 students	Questionnaire	Software engineering management Software engineering process	Knowledge	Learning was not assessed.
GS-02	¹ Quantitative Modeling for the Interactive Simulation of Software Project (A. Drappa & Ludewig, 1999) ² Simulation in Software Engineering (Anke Drappa & Ludewig, 2000)	SESAM (Software Engineering Simulation by Animated Models) is a model of a software project. Users run the model loaded with its initial project state and then tweak it to simulate different scenarios before running it again. Players take the role of a project manager and must plan and control a simulated project. Rather than a graphical user interface, players control the game by typing commands in a modelling language. Players analyse their performance through an after-game analysis tool.	¹ Non-experimental ² True Experimental	¹ 10 undergraduate project management students ² 19 second-year computer science students	¹ n/a ² Pre- and post-game tests Project plan	Software engineering management	Knowledge	¹ A qualitative assessment that the players experienced something similar to a real project, including panic when the deadlines were approaching. ² Students in the experimental and control groups improved their performance in successive game sessions.
GS-03	An Interactive Multimedia Software House Simulation for Postgraduate Software Engineers (Sharp)	Case studies are presented through a simulated office environment and then completed outside of the game environment.	Non-experimental	Post-graduate distance education software engineering students.	Questionnaire	Software requirements Software design Software construction Software	Knowledge	Learning was not assessed.

	& Hall, 2000)					testing		
GS-04	How to Successfully Use Software Project Simulation for Educating Software Project Managers (Mandl-Striegnitz, 2001)	Participants play two sessions of SESAM (GS-02) and their tutor analyzed their performance and provided feedback in between.	Non-experimental	40 undergraduate software engineering students	Questionnaire	Software engineering management	Knowledge	Players improved their performance in the second session but still had problems monitoring their project and tracking progress.
GS-05	An Experiment for Evaluating the Effectiveness of Using a System Dynamics Simulation Model in Software Project Management Education (D. Pfahl, Koval, & Ruhe, 2001)	A three-phase (design, implementation, test) waterfall project modeled using System Dynamics. Key project variables were project duration, effort consumption, product size, and quality after testing. Participants were separated in two groups: one group managed a simulated software project with the aid of a System Dynamics model (Abdel-Hamid, 1989); the other group used COCOMO (Boehm, Abts, Brown, Chulani, Clark, Horowitz, Madachy, Reifer & Steece, 2000).	True Experimental	12 post-graduate software engineering students	Pre- and post-test questionnaires	Software engineering management	Knowledge	Pre- and post-session surveys indicated that participants were improving their knowledge of project management patterns and behaviors. Those using the simulation models performed better than those using COCOMO.
GS-06	An Externally Replicated Experiment for Evaluating the Learning Effectiveness of Using Simulations in Software Project Management Education (Dietmar Pfahl, Laitenberger, Dorsch, & Ruhe,	Same as for GS-05.	True Experimental	¹ 12 graduate and post-graduate students majoring in computer science. ² 13 senior undergraduate students majoring in computer	Pre- and post-test questionnaires	Software engineering management	Knowledge	The results confirmed the initial findings in which students using the System Dynamics model generally performed better in the pre- and post-test questionnaires than those using COCOMO.

	<p>2003)</p> <p>Evaluating the Learning Effectiveness of Using Simulations in Software Project Management Education: Results From a Twice Replicated Experiment (Dietmar Pfahl, Laitenberger, Ruhe, Dorsch, & Krivobokova, 2004)</p>			<p>science, electrical engineering, and computer engineering.</p>				
GS-07	<p>Problems and Programmers: An Educational Software Engineering Card Game (Baker, Navarro, & van der, 2003)</p> <p>An Experimental Card Game for Teaching Software Engineering Processes (Baker, Oh Navarro, & van der Hoek, 2005)</p> <p>Teaching Software</p>	<p>A competitive card game called Problems and Programmers in which students play the role of project manager in a waterfall project. All players lead the same project. Players must balance several competing concerns including budget and the client's demands regarding the reliability of the final software. Who finishes first, wins.</p>	<p>Non-experimental</p>	<p>28 undergraduate students who had completed an introductory software engineering unit</p>	<p>Questionnaire</p>	<p>Software engineering managementS oftware engineering process</p>	<p>Knowledge</p>	<p>Players self-assessed their level of learning in a post-game survey. Most said the game was not good at teaching new knowledge or reinforcing existing knowledge.</p>

	Engineering Using Simulation Games (Navarro, Baker, & van der Hoek, 2004)							
GS-08	Engendering an Empathy for Software Engineering (Shaw & Dermoudy, 2005)	Players act as a project manager to deliver a product within time and budget constraints. SimjavaSP uses discrete-event simulation as the game engine.	Non-experimental	Undergraduate software engineering students	Post-test questionnaire	Software engineering management	Knowledge	The degree of learning was self-assessed by the participants and was found to be positive.
GS-09	Model-Driven Game Development: Experience and Model Enhancements in Software Project Management Education (Barros, Dantas, Veronese, & Werner, 2006) A Simulation-Based Game for Project Management Experiential Learning (Dantas, Barros, & Werner, 2004)	Uses simulation to support decision-making on software project management. In the game, The Incredible Manager, the player sets project parameters such as staffing and work hours and executes the project for a period of time. The simulation can be stopped so the parameters can be tweaked.	Non-experimental	7 post-graduate students in a software project management course, 8 undergraduate and post-graduate students from a software development laboratory, 9 other undergraduates.	Questionnaire	Software engineering management	Knowledge	Players self-assessed their level of learning in a post-game survey. Most said they had learned something new but only one person completed their project successfully.
GS-10	SimVBSE: Developing a Game for Value-Based Software Engineering (Jain & Boehm, 2006)	Focused on value-based software project management: every requirement, use case, object, test case and defect is treated as equally important; earned value is used to track project cost and schedule; a separation of	n/a	n/a	n/a	Software engineering management	Knowledge	n/a

		concerns is practiced, in which the responsibility of software engineers is confined to turning software requirements into verified code. The player's avatar visits different game rooms and collects information from stakeholders about the current project and how to proceed.						
GS-11	SimSE: A Software Engineering Simulation Environment for Software Process Education (Navarro, 2006).	Same as for GS-14.	True Experimental	19 undergraduate software engineering students	Pre- and post-test questionnaires	Software engineering management Software engineering process	Knowledge	All groups improved their knowledge, but those in the control groups outperformed those who had played SimSE in the post-test. When players play SimSE for longer periods, their scores improved. But, many dropped out due to boredom or frustration before this point.
GS-12	e-Learning in Project Management Using Simulation Models: A Case Study Based on the Replication of an Experiment (Rodriguez, Sicilia, Cuadrado-Gallego, & Pfahl, 2006)	A replication of the GS-05	True Experimental	11 second-year undergraduate students taking a software engineering module	Pre- and post-test questionnaires	Software engineering management	Knowledge	According to the post-test and qualitative results, students using the simulation appear to have understood the software engineering principles it was trying to teach better than those in the control group
GS-13	Using Games in Software Engineering Education to Teach Risk	A competitive board/card game that focuses on risk management. Players take the role of project manager and have to develop a product and sell it in the market.	Non-experimental	150 on-campus and distance students.	5-question questionnaire	Software engineering management	Knowledge	Players said they understood the learning objectives of the game. The degree of learning was not assessed.

	Management (Taran, 2007)	The player with most money at the end wins. A dice is used to simulate eventuated risk events.						
GS-14	<p>Towards Game-Based Simulation as a Method of Teaching Software Engineering (Oh & van der Hoek, 2002)</p> <p>Design and Evaluation of an Educational Software Process Simulation Environment and Associated Model (Navarro & van der Hoek, 2005)</p> <p>SimSE: A Software Engineering Simulation Environment for Software Process Education (Navarro, 2006)</p> <p>Comprehensive Evaluation of an Educational Software Engineering Simulation Environment (Navarro & van</p>	<p>A single-player game for multiple development methodologies (waterfall, RUP, rapid prototyping) in which the player takes the role of a project manager leading a team of developers. The team must complete a virtual software project by hiring staff, assigning tasks, monitoring progress, purchasing resources.</p> <p>At the end of the game the player receives a score and can analyse their results with an explanatory tool.</p>	Non-experimental	29 undergraduate software engineering students	Post-test questionnaires	Software engineering management Software engineering process	Knowledge	<p>Players felt the game reinforced what they already knew but provided little new knowledge.</p> <p>Players are demonstrating aspects of learning theories such as learning by doing, situated learning, discovery learning, learning through failure, and Keller's ARCS.</p> <p>SimSE is most effective when used with other teaching methods.</p>

	der Hoek, 2007)							
GS-15	Enhancing Software Engineering Education Using Teaching Aids in 3-D Online Virtual Worlds (Ye, Chang, & Polack-Wahl, 2007)	Two exercises were performed in Second Life, an online virtual environment. ¹ Groupthink exercise: groups of students are given a software specification and must reach a design consensus. Afterwards, individuals are asked questions about the specification and points are awarded for correct answers. ² SimSE exercise: the game from GS-14 was modified to run in Second Life.	Non-experimental	¹ 29 undergraduate students ² 26 undergraduate students	Questionnaire	Software engineering process Software requirements Software engineering management	Comprehension	Most students said the exercises helped them understand the fundamentals of software specification activities and the principles of software development processes.
GS-16	Requirements Game: Teaching Software Project Management (Zapata & Awad-Aubad, 2007)	Teams of 4 or 5 players take on roles such as project manager, developers, designers, or analysts. For a given case-study, the players must produce documentation such as an ER diagram, sketches of at least 3 GUIs, and an estimation of the effort required, and then build the application in, say, Microsoft Access. A facilitator plays the role of a client giving more instructions or clarifications. Fines may be imposed for time or budget over-runs.	Non-experimental	47 systems engineering undergraduate students. 8 systems engineering Masters students. 30 systems, industrial, and administrative engineering undergraduate students.	Performance in the game alone	Software requirements	Knowledge	Not assessed
GS-17	A Game for Taking Requirements Engineering More Seriously (Knauss, Schneider, & Stapel, 2008)	A web-based game that can be completed in about 10 minutes. Software requirements are visualized as a bag of balls that flow from the customer to an analyst, a designer, and a developer depending on the development process chosen. Alternate flows may be taken (such as the client speaking	n/a	n/a	n/a	Software requirements	Knowledge	Not assessed

		directly to the developers to clear up misunderstandings), which can change the rate of flow.						
GS-18	On the Role of Learning Theories in Furthering Software Engineering Education (Navarro & van der Hoek, 2008)	Same as for GS-14.	Quasi-experimental	11 undergraduate students who had passed an introductory software engineering course.	Observation and post-test interview	Software engineering management Software engineering process	Knowledge	Players demonstrated aspects of learning theories such as learning by doing, situated learning, elaboration, discovery learning, learning through failure, Keller's ARCS, and learning by reflection.
GS-19	An Evaluation of a Mobile Game Concept for Lectures (A. I. Wang, Fsdahl, & Morch-Storstein, 2008)	The lecturer acts as a game show host and students answer multiple choice questions about a particular software design issue through their laptop or mobile phone. Players have to answer correctly to get to the next round. The winner is the last person standing.	Non-experimental	20 software engineering Masters students.	Questionnaire Performance in the game	Software design	Knowledge	Players felt the system made them pay closer attention during the lecture and that they learned more than through a traditional lecture.
GS-20	Multi-Site Evaluation of SimSE (Navarro & van der Hoek, 2009)	Same as for GS-14. SimSE was run in game sessions in which the original game designers were not directly involved.	True Experimental	Site 1: 14 students in a senior research seminar course, most of whom had passed a software engineering course. Site 2: 19 undergraduate software engineering students. Site 3: 48 under-	Post-test questionnaires, performance in SimSE, and final course grades.	Software engineering management Software engineering process	Knowledge	Students seemed to learned the concepts the game is designed to teach. The game was suitable for students of varying abilities and backgrounds. SimSE is most effective when used with other teaching methods.

				graduate software engineering students.				
GS-21	Empirical Evaluation of an Educational Game on Software Measurement (Gresse von Wangenheim, Thiry, & Kochanski, 2009)	In X-MED, the player takes the role of a measurement analyst and defines and executes a measurement exercise based on a given development scenario. A score is calculated based on the number of correct decisions made, and the player is presented with an analysis of their performance.	True Experimental	15 computer science post-graduate students	Pre- and posttest questionnaires	Software engineering management Software engineering process	Knowledge	The results don't conclusively point to a positive learning effect, although most players' subjective evaluation was that the game helped them understand the topic.
GS-22	Adapting Game Technology to Support Software Engineering Process Teaching: From SimSE to MO-SEProcess (Zhu, Wang, & Tan, 2007) A Software Engineering Education Game in a 3-D Online Virtual Environment (T. Wang & Zhu, 2009)	A game based on SimSE (GS-14) using the rapid prototyping profile and deployed to Second Life.	Non-experimental	52 software engineering students	A six-question post-test questionnaire.	Software engineering process	Knowledge	Players self-assessed their level of learning in a post-game survey. Most said the game had helped them understand the software development process better.
GS-23	PlayScrum- A Card Game to Learn the Scrum Agile Method (Fernandes & Sousa, 2010)	Focused on the Scrum (Schwaber, 2004) agile development process. Further development of Problems and Programmers (Baker et al., 2005). Played by 2 to 5 people. Cards are used to represent tasks,	Non-experimental	13 post-graduate students.	Questionnaire	Software engineering management Software engineering process	Knowledge	Students improved their performance in successive game sessions. Players analyze their performance through an after-game analysis tool

		problems, developers, and artifacts. The winner is the person who performs all tasks with the least number of errors. A roll of a dice determines the flow of the game.						
GS-24	<p>Evaluation of a Game to Teach Requirements Collection and Analysis in Software Engineering at Tertiary Level (Hainey, Connelly, Stansfield, & Boyle, 2010)</p> <p>An Application of Games-Based Learning Within Software Engineering (Connolly, Stansfield, & Hainey, 2007)</p>	<p>Players take on specific roles (project manager, systems analyst, systems designer, team leader). The systems analyst moves their avatar through the game world to collect requirements by asking questions of game characters. When the analyst thinks they have all requirements, they prepare a requirements document and send it to the project manager, who must decide whether to proceed with the project.</p>	True Experimental	55 university students and 37 higher-education students (92 in total). The majority had little or no instruction in requirements collection or analysis.	Pre- and post-test questionnaires	Software requirements	Knowledge	<p>Comparison of pre- and post-game test scores showed an increase in knowledge. Control groups who did not play the game also showed an increase in knowledge. The game was found to be a good supplement to existing courses. Higher education students gained more from the game (better post-game scores) and were more accepting of the teaching technique than further education students.</p>
GS-25	Learning Software Engineering Basic Concepts Using a Five-Phase Game (Rusu, Russell, Robinson, & Rusu, 2010)	<p>Players take the role of a requirements engineer in a waterfall development (requirements, design, implementation, testing, maintenance phases) software project. The player's avatar must ask questions of on-screen characters to determine the right requirements. Subsequent phases use arcade-style graphics to kill 'computer bugs' or to 'shoot'</p>	Non-experimental	Developed by teams of undergraduate software engineering students and used by a class of nine middle and high school students with	Pre- and post-test questionnaires	Software engineering management	Knowledge	<p>Comparing pre- and post-game surveys most participants said they gained a better understanding of software development.</p>

		answers in a multiple choice quiz.		limited or no computer science background.				
GS-26	A Classroom Game for Teaching Management of Software Companies (Zapata, 2010)	Players take turns in rolling a dice and answering a technical question about software development. If the answer is right, the player's team has the chance to solve a project estimation problem. The team with the most correct responses to the questions and estimation problems wins.	Non-experimental	40 systems engineering students	Post-test questionnaire	Software requirements	Knowledge	Players self-assessed their level of learning in a post-game survey. Most said they had learned something new.

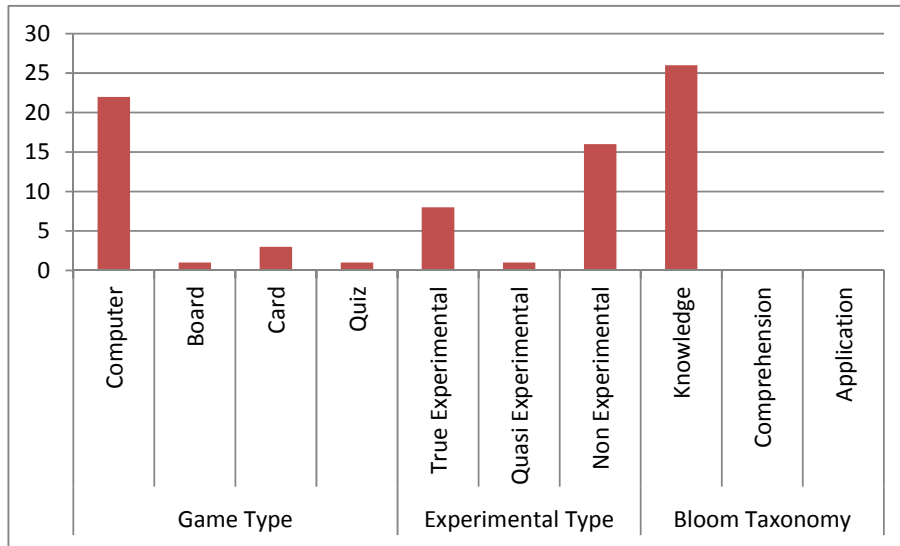
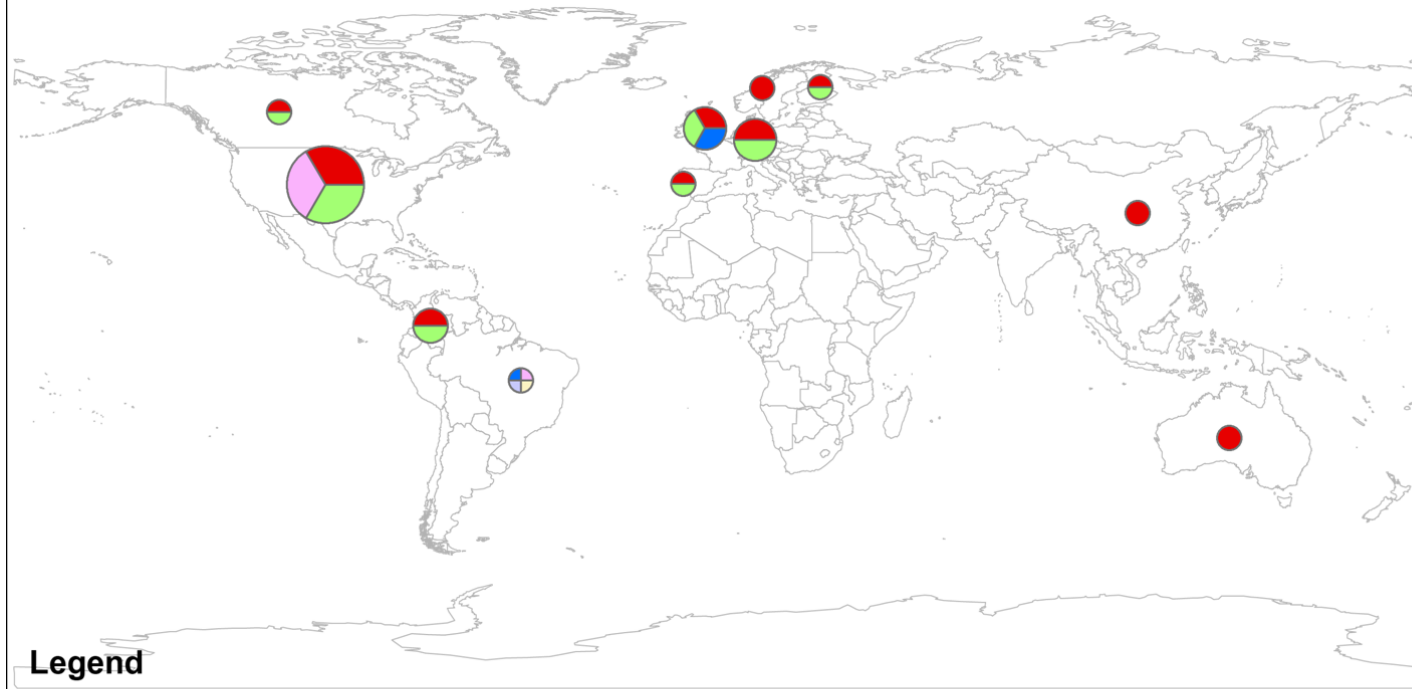


Figure 1. Game surveys classified by game type, experimental type, and Bloom taxonomy.

Games Used for Software Engineering Management Education



Legend

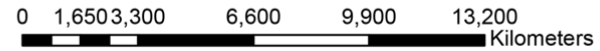


Figure 2. Games used for software engineering education by location and SWEBOK knowledge area

Shall We Play a Game?

Craig Caulfield (Corresponding author)

School of Computer Science and Security Science, Edith Cowan University
2 Bradford Street, Mount Lawley, Western Australia 6050, Australia
Tel: 61-8-9370-6295 E-mail: ccaulfie@our.ecu.edu.au

S P Maj

School of Computer Science and Security Science, Edith Cowan University
2 Bradford Street, Mount Lawley, Western Australia 6050, Australia
Tel: 61-8-9370-6277 E-mail: p.maj@ecu.edu.au

Jianhong (Cecilia) Xia

Department of Spatial Sciences, Curtin University
Kent Street, Bentley, Western Australia 6102, Australia
Tel: 61-8-9266-7563 E-mail: c.xia@curtin.edu.au

D Veal

School of Computer Science and Security Science, Edith Cowan University
2 Bradford Street, Mount Lawley, Western Australia 6050, Australia
Tel: 61-8-9370-6295 E-mail: d.veal@ecu.edu.au

Received: October 8, 2011

Accepted: October 19, 2011

Published: January 1, 2012

doi:10.5539/mas.v6n1p2

URL: <http://dx.doi.org/10.5539/mas.v6n1p2>

Abstract

This paper presents the results of a qualitative research project that used a simple game of a software project to see if and how games could contribute to better software project management education, and, if so, what features would make them most efficacious. The results suggest that while games are useful pedagogical tools and are well-received by players, they are not sufficient in themselves and must be supplemented by other learning devices.

Keywords: Software engineering, Project management education, Serious games

1. Introduction

In the 1983 movie, *War Games*, a young Matthew Broderick plays David Lightman, a hacker who has broken into WOPR– the War Operation Plan Response supercomputer which is programmed to play out different doomsday scenarios and learn from them so it can eventually take full, automated control of the United State’s nuclear arsenal. When David is presented with a screen prompt that asks, “Shall we play a game?”, he innocently selects “Global Thermonuclear War”. As quickly becomes apparent, WOPR is ready to do more than just play games and it starts executing commands in readiness for a real missile strike against the Soviet Union.

The portentous question asked by WOPR– shall we play a game?– has meaning for the research project discussed in this paper too, but without the same dire consequences. This paper reports on a qualitative research project designed to see if and how games could contribute to better software project management education by helping software engineers and project managers explore some of the dynamic complexities of the field in a safe and inexpensive environment. If games could indeed contribute, then what features made them most efficacious? Games have been used to good effect in other similarly dynamic areas and the researchers believed that an opportunity existed to see what contribution they could make to a better software project education. In effect:

shall we— should we— play games in software project management education?

2. Literature Review and Background

2.1 Defining Games

To play a game, “is to engage in activity directed towards bringing about a specific state of affairs, using only means permitted by specific rules, where the means permitted by the rules are more limited in scope than they would be in the absence of the rules and where the sole reason for accepting such limitation is to make possible such activity” (Suits, 1967, p.156).

A game is different from a model or simulation. For example, a model is “a miniature representation of a complex reality. A model reflects certain selected characteristics of the system it stands for. A model is useful to the extent that it portrays accurately those characteristics that happen to be of interest at the moment” (DeMarco, 1982, p.41). Meanwhile, a simulation is a special kind of model that shows how a system, such as a biological, social, physical, or economic system, changes over time (Miller, 1978, p.83).

The games that this research project deals with are a subset of Caillois’s (1961) *agôn* classification, those games “that would seem to be competitive like a combat in which equality of chances is artificially created in order that the adversaries should confront each other under ideal conditions” (Caillois, 1961, p.14), and they use an adjective— serious— to show they are not for simple amusement and they are designed to educate, train, or inform their players (Abt, 1970; Michael & Chen, 2005; Schrage & Peters, 1999).

2.2 The Nature of Software

Software development is an inherently complex endeavour because of both the ephemeral qualities of software itself and the dynamic socio-technical system in which it is developed. For example, software has no fundamental theory (Osterweil, 1987, p.3), like the law of physics, with which we can reason about its behaviour. This makes it difficult to thoroughly test software without actually building it and running it in a live environment (Kruchten, 2005) with all the attendant risks this involves.

Software must also conform to the arbitrary design of the human institutions and processes in which it is deployed and accept change because in a system of software, hardware, and humans, it is the most malleable (Brooks, 1987, p.12). These are naturally properties that organisations want to take advantage of, but constant change, if not managed, can erode the integrity of the original design, and when combined with relatively low manufacturing costs, can lead to shortcuts:

Program implementation is more like preparing a cast in mechanical engineering. The real “manufacturing” of software entails almost no cost; a CD-ROM, for example, costs less than a dollar, and delivery over the Internet only a few cents. Often it doesn’t matter if the design is a bit wrong; we can just fix it and manufacture it again. You can’t do that with a bridge or a car engine because the cost would be huge, and that forces engineers involved in building these things to get them right the first time (Kruchten, 2004).

Because software is complex, difficult to reason about and test, and yet cheap and easy to change, it is perhaps understandable that many implementations are not right the first time, if at all. These qualities of software are often the root cause of many quality and productivity issues:

From the complexity comes the difficulty of communication among team means, which leads to product flaws, cost overruns, schedule delays. From the complexity comes the difficulty of enumerating, much less understanding, all the possible states of the program, and from that comes the unreliability. From the complexity of the functions comes the difficulty of invoking those functions, which makes programs hard to use. From complexity of structure comes the difficulty of extending programs to new functions without creating side effects. From complexity of structure comes the unvisualized states that constitute security trapdoors (Brooks, 1987, p.11).

How then do we prepare software engineers to work in an environment that is complex in and of itself, and which is, in turn, used to create a complex product? To answer this we need to look at the state of current professional practice and the educational programs that produce new software engineers.

2.3 The Education of Software Engineers

In response to some of the productivity and quality problems in the field, steps are being taken to make software engineering more reliable, more predictable, more like its engineering namesake:

A body of knowledge, the SWEBOK (Bourque, Dupuis, Abran, Moore, & Tripp, 1999), has been defined which captures accepted practice in the field and which also forms the basis of curriculum development and

accreditation, licensing and certification programs.

Standards of ethics and conduct have been developed to guide software engineers in responsible behaviour, although these are still optional and unenforceable (Gotterbarn, 1999; McConnell, 2004, p.57).

Professionally-endorsed curriculum guidelines for graduate and post-graduate software engineering education have been developed to meet the latest technical developments and evolving industry demands. These include Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering (Joint Task Force on Computing Curriculum, 2004), Curriculum Guidelines for Graduate Degree Programs in Software Engineering (iSSEc Project, 2009), and Curriculum Guidelines for Undergraduate Degree Programs in Information Systems (Joint IS2010 Curriculum Task Force, 2010).

Of interest for this research project was the way software engineers are educated because this directly and significantly affects so many other areas of professional practice. In the curriculum guidelines considered mentioned above, each identifies better software project management skills and better soft, or peopleware (DeMarco & Lister, 1999), skills as critical for all graduating students, but the guidelines are intentionally vague on how institutions should teach these.

This pedagogical gap is exposed most often when students finish their requisite courses and attempt their final, important, and synthesising capstone project. While there are many cases of capstone projects bringing great benefits for the students and their clients (Boehm, et al., 1998; Johns-Boast & Patch, 2010), these are balanced by stories of significant failures in which student/client relationships broke down, there was severe internal team dissension, or the final software was unusable (Brereton, et al., 2000; Cheng & Lin, 2010; Polack-Wahl, 2006). For those capstone projects that failed, there was little opportunity for reflection or remedial action because the project was the final unit of study for the course. Some research has been conducted that recommends guidelines for successful capstone projects (Robillard & Robillard, 1998), such as providing students with basic training in project control, reviewing the design documents, and having an experienced software engineer mentor certain stages, but these are relatively costly or time-consuming course attributes and there is little evidence they have been widely adopted.

2.4 One Possible Solution

One possible way to tackle these problems is to use a serious game— a game designed to teach and educate players about some of the dynamic complexities of the field in a safe and inexpensive environment. Importantly, games are not one-shot opportunities in the way capstone projects are: a game can be played, studied, tweaked, and replayed as many times as needed.

Games have been used in many different business (Michael & Chen, 2005; Schrage & Peters, 1999), military (Perla, 1990; Riddell, 1997; Zyda, 2007), and social environments (Gee, 2007; Prensky, 2006; Salen & Zimmerman, 2005), and have proven to be efficacious. They also draw their intellectual integrity from sound education theory such as problem-based learning. Problem-based learning is a pedagogic method that uses problem scenarios to encourage students to work out solutions for themselves (Barell, 2006; Barrows & Tamblyn, 1976; Savin-Baden, 2003; Savin-Baden & Major, 2004). Usually working in small teams, students explore the problem, bring their personal experience to bear, identify any gaps in their knowledge, and eventually come up with viable solutions. The problems themselves are usually complex, ill-defined, real-world situations for which there may not necessarily be a single right or wrong solution (Maxwell, Mergendoller, & Bellisimo, 2004, p.2). The students build new knowledge through self-directed learning while their tutors act as facilitators or consultants rather than more traditional instructors (Dempsey, Haynes, Lucassen, & Casey, 2002, p.5; McCall, 2011). Games therefore have an authority to be taken seriously as learning and research tools.

This research project is not the first to look at games in software engineering education. A systematic survey of the field (Caulfield, Xia, Veal, & Maj, 2011) discovered over two dozen research projects using mostly single-user computer games to teach various aspects of the software development lifecycle. However, few of these games were developed or repeated beyond their initial implementations, which suggests that their design lacked some essential feature.

An opportunity therefore existed to explore more fully if and how games could contribute to better software engineering management and help fill some of the pedagogical gaps in the current curriculum guidelines; and if they could, then what design features were most valuable.

3. Methodology

3.1 Introducing Simsoft

The primary research tool for this project was a game called Simsoft (Caulfield, Veal, & Maj, 2011a). Physically, Simsoft comes in two pieces. There is an A0-sized printed game board around which the players gather to discuss the current state of their project and to consider their next move. The board shows the flow of the game while plastic counters are used to represent the staff of the project. Poker chips represent the team's budget, with which they can purchase more staff, and from which certain game events may draw or reimburse amounts depending on decisions made during the course of the game. There is also a simple Java-based dashboard (Caulfield, Veal, & Maj, 2011b), through which the players can see the current and historical state of the project in a series of reports and messages; and they can adjust the project's settings. The engine behind Simsoft is a system dynamics model which embodies the fundamental causal relationships of simple software development projects.

3.2 Game Sessions

Simsoft game sessions were held between May and September 2010 in which teams of students, and practicing project managers and software engineers managed a hypothetical software development project with the aim of completing the project on time and within budget (with poker chips left over). Based on the starting scenario of the game, information provided during the game, and their own real-world experience, the players made decisions about how to proceed— whether to hire more staff or reduce the number, what hours should be worked, and so on. After each decision set had been entered, the game was run for another next time period, (a week, a month, or a quarter). The game was now in a new state which the players had to interpret from the game board and decide how to proceed.

3.3 Participants

Purposive sampling (Lincoln & Guba, 1984; Patton, 2002) was used to select the participants (n=59) of the study from the following pools:

Post-graduate project management students from two Perth, Western Australia Universities.

Software engineers, project managers, and account managers from a Perth-based software consulting company.

A call for participation was distributed by email and the participants replied if they wished to take part. Snowball sampling (Marshall, 1996) was allowed, whereby those reading the email were encouraged to refer others in the same field they thought would be interested in taking part.

Although the participants each had an information technology or project management background, they exhibited notable variances in experience (from recent graduates to 25-year industry veterans); skills (from those still studying to highly-certified professionals); and cultural diversity (the participants came from Australia, Europe, the Middle East, Asia, and South Africa).

3.4 Data Collection

Before the game session, the players completed an online survey designed to test their knowledge of general software engineering and project management principles. The survey questions were based on those in examination preparation guides for the IEEE's Certified Software Development professional certification (Naveda & Seidman, 2006) and the Project Management Professional certification (Heldman, 2007).

After the game session, the players completed another online survey. Post-game surveys are a common feature of game research (Eldredge & Watson, 1996; Faria, 1987, 1998; Faria & Wellington, 2004; Faria & Wellington, 2005; McKenna, 1991) and in problem-based learning (Tang, et al., 1997), the key foundation of Simsoft's design. Based on these exemplars, a survey was designed to capture their experience of playing the game, whether they found it useful, how it might compare to other forms of instruction such as lectures or case studies, and what may have been learned through the game.

Therefore, this research project had multiple data sources: the Simsoft game database, the pre- and post-game surveys, interviews with the players, researcher memos (Maxwell, 2004), and field notes.

3.5 Data Analysis

The analysis and interpretation of the data for this project followed a path used many times before in qualitative research: collect the data, analyse it for themes or perspectives, and then report on four or five of those themes (Bloomberg & Volpe, 2008; Creswell, 2009; Lincoln & Guba, 1984). In more detail, the following steps were taken:

Organized and prepared the data for analysis by transcribing the interviews, and writing up the field notes and memos.

Re-read, examined, and explored the data to get a high-level sense of what had been collected.

Started to analyse the data by first coding it— breaking it into named chunks or categories that can then be used to make comparisons between things in the same category and to help develop theoretical concepts (Rossman & Rallis, 1998; Strauss, 1987). The software package NVivo (<http://www.qsrinternational.com/>) was used for this task (Richards, 2009).

Used the coding to identify and describe themes and patterns in the data, which then became candidates for more detailed analysis and, potentially, major findings of the project (Maxwell, 2004).

Formulated the finding statements and supported these with specific data instances and then summarized the key findings.

Sought meaning in the findings by linking it to experience, insight, or the literature. The most commonly asked question was: “What were the lessons learned?” (Lincoln & Guba, 1984).

The above step-by-step list might give the impression that the analysis and interpretation of the data proceeded in a linear fashion once all the data had been collected. In reality, the process was highly iterative and started as soon as the first data was available— a feature common to this type of research (Lincoln & Guba, 1984, pp.241-242).

3.6 Reliability, Validity, and Applicability of the Findings

Compared to objective, deductive quantitative research, qualitative research has often been called undisciplined, sloppy, merely subjective, and indiscriminately responsive to the loudest bangs and brightest lights (Lincoln & Guba, 1984, p.289). Add to this a researcher intimately involved in the data collection and carrying certain biases, and it is natural to question the trustworthiness of the results: “The basic issue in relation to trustworthiness is simple: How can an inquirer persuade his or her audiences (including self) that the findings of an inquiry are worth paying attention to, worth taking account of? What arguments can be mounted, what criteria invoked, what questions asked, that would be persuasive on this issue?” (Guba & Lincoln, 2005, p.290). That is, how can we demonstrate the findings are reliable, valid, and applicable?

For this project the following means were used:

Reliability, or “the extent to which a measurement procedure yields the same answer however and whenever it is carried out” (Kirk & Miller, 1986, p.19): the coding scheme peer reviewed by an independent party.

Validity, or the “degree to which the finding is interpreted in a correct way” (Kirk & Miller, 1986, p.20): more than data source was used so that the results could be triangulated; and the findings were reviewed by a random sample of four players from the games sessions.

Applicability, or “the extent to which the findings of a particular enquiry have applicability in other contexts or with other subjects” (Guba & Lincoln, 2005, p.290): rich, detailed descriptions are provided that will allow subsequent researchers to determine if the findings are relevant to their particular setting.

4. Findings

Six major findings emerged from the research.

4.1 Finding 1— *There was evidence the participants were learning by doing.*

A key tenet of problem-based learning (Savin-Baden & Major, 2004), one of the theoretical foundations of Simsoft, is that when people work through problems for themselves, the knowledge they build ‘sticks’ and they are more able to apply what they have learned in new situations. The following comments indicate that playing Simsoft indeed helped the participants figure things out for themselves:

“Aha!”

“Our team figured out we could move more counters [work units] by investing in a couple of expensive, experienced developers, more middies, and some quality control people. Makes sense really”

“We spent our poker chips on lots of cheap newbies and before long had most of our counters [work units] in rework. We should have bought some old timers for guidance”

“Now I see why”

“I hadn't appreciated the level of productivity variability between developers before”

In addition, all participants completed pre- and post-game surveys that included a number of questions designed to test their general level of knowledge about project management and software engineering concepts. Table 1 shows the results of the tests broken by the participants' role and years of experience, and shows each group performed better after playing the game. Two non-parametric statistical tests were run over the pre- and post-game results to determine if this improved performance was significant.

A Mann-Whitney U test ($Z = -1.091, p = 0.275 > 0.05$) indicated that there was no significant differences between the pre- and post-game results when considering the broad groups of project managers, software developers, and students. A Wilcoxon signed ranks test ($Z = -1.604, p = 0.109 > 0.05$) also showed there was no significant difference between the pre- and post-game results of the three groups.

The same statistical tests were then run at a finer level of detail: against the years-of-experience sub-groups with the three main groupings of project managers, software developers, and students. Both the Mann-Whitney U test ($Z = -2.951, p = 0.003 < 0.05$) and the Wilcoxon signed ranks test ($Z = -2.552, p = 0.011 < 0.05$) showed there was a significant improvement between the pre- and post-game tests.

Together, these results indicate that while playing the game helped, none of the three main groups performed significantly better than the others. However, the years of experience a person has may affect how much they take from the game.

4.2 Finding 2— Games such as Simsoft are not sufficient learning vehicles by themselves and need to be supplemented by other methods.

While most players (40 out of 59) said that Simsoft helped put project management and software engineering theories into a practical context, the mean score was 2.64 out 5 (SD = 0.760) when they were asked if games were a better way of learning and understanding technical material than through more conventional methods such as books, lectures, case studies.

From an experienced software developer:

"I saw in the game aspects of theory covered at uni, but without knowing the theory first I probably wouldn't have recognised the significance."

And these comments from two students:

"I was out of my depth"

"I could see the logic behind my team's decision, but I wouldn't have known enough to make the decision by myself."

One project manager expressed an interest in using Simsoft as part of an under-graduate computer science he teaches part-time, but:

"It would have to be used on the final weeks of the course when the students have some theory under their belt. Plus, there is little momentum behind problem-based learning at [my university] so the resources aren't available to design a proper PBL based curriculum"

Table 1 also shows that the greatest improvement between the pre- and post-game tests was in those groups with the greatest work experience, so that relatively inexperienced participants took less from the game. This suggests that some level of *a priori* knowledge is needed for games like Simsoft to be truly effective.

However, when asked if games were a better way of more thoroughly learning a topic than through more conventional methods such as books, lectures, case studies, a significant minority (21 out of 59 participants) agreed or strongly agreed (mean = 3.00 out of 5, SD = 0.964). Self discovery seems to be the motive:

"I like to figure things out for myself"

On six occasions over the seven game sessions, the researcher overheard players saying they wished they could set Simsoft to match their work environment so they could game through some current issues.

4.3 Finding 3— Simsoft is a suitable pedagogical device for participants of different skills and backgrounds.

When asked if the game was easy or hard to play (1 = too easy, 3 = about right, and 5 too hard), the majority of the participants (47 out of 59) felt that the game was pitched at about the right level of difficulty (see Table 2).

This comment was from a student:

"Even though I'm still studying and don't have much [practical work] experience, I was able to understand the game's project and contribute to the decisions"

And, from a project manager with 10 to 15 years experience:

"[The] game was not too easy so that it was boring, but not too hard that newbies couldn't undetstad (sic) it."

Across the seven game sessions there were no teams composed entirely of one group only, so each had a mixture of skills and experience. This was viewed positively:

"Our team had a mixture of abilities and life experience. I think this helped us make good choices"

"[One of our team] had read about Brooks' model and could let us know if we were on the right track"

4.4 Finding 4— The majority (49 out of 59) of participants said they would be prepared to invest greater time and effort in games such as Simsoft if the reward was deeper understanding of a problem domain.

Many players said they reached the end of the game before they had time to fully explore the dynamics of the scenario, or they wanted to take more discussing their options before committing to a decision. For example:

"The game was too short to discover what I wanted to know"

"I wanted to know more"

"We wanted more time to talk about our options"

The database of Simsoft game transactions showed that games lasted an average of 35 minutes (SD = 7.082) and that 80% of games finished within 40 minutes. The players were encouraged to stay behind after the game sessions to discuss and compare their results with other teams. Often, these after-game sessions lasted longer than the games themselves.

Considering the amount of time they had spent playing Simsoft, a majority of the players (49 out of 59) said would be prepared to invest greater time and effort in games like Simsoft if the reward was greater understanding of the problem domain:

"What about running the game in real time, like the stock market game. That would give us time to make really considered judgements, people could be assigned research topics during the week"

"I hope that future versions will let me set up specific scenario and play them out. That would really help me in my work"

Outside of this research project, 10 players had previously participated in a long-running online stock market game in which notional shares were bought and sold based on actual prices published in a daily newspaper. Buy and sell decisions were submitted weekly and the team with the largest portfolio after three months was declared the winner.

4.5 Finding 5— The majority (44 out of 59) of the participants found that working in groups was a positive experience

An important component of many of the pedagogical theories behind Simsoft is the aspect of working in groups or teams, so it was important to assess how this was received by the players. A majority of players (44 out of 59) said they found it useful or very useful to work as a team and that this reflected how things often happened in the workplace:

"It was like [the agile] stand up meeting we have every morning"

"We organised ourselves into roles we felt comfortable with or that fitted our day-job: someone on the calculator, someone moving the developer pieces, someone moving the units of work"

However, one student found something new in the practice:

"I thought software development was a solitary experience but it's not really"

Others liked the opportunity to share opinions and learn from more experienced peers:

"Everyone had a chance to offer an opinion"

"I have little real-world project experience so it was good to get the advice of others and see how they approached problems"

But, as in any group activity, the game facilitator needs to be aware of cultural differences that may make some less inclined to contribute and of players who are dominating in their groups:

"Generally, everyone had their say in final decision but a couple of times we were overridden"

4.6 Finding 6— The majority (44 out of 59) of participants preferred playing a board game rather than a fully computerized game

The players' responses to different features of the game were generally positive (Table 3). Notable in Table 3 is that a majority of players (44 out of 59) preferred playing with a game board rather than a fully computerized version. Some typical comments were:

"The board game [was] simple and I could easily see the state of the game"

"When a group plays the game on a PC, someone controls the mouse and keyboard and they tend to dominate"

"Compared to computer-based games, the design was simple and we started playing without too much wasted time"

"Sometimes technology gets in the way"

"Everyone plays board games so we all knew what to do"

Outside of this research project, seven players had played The Beer Game, four-point distribution chain, originally developed at MIT and now used widely as a management educational tool in a variety of academic and commercial settings (Caulfield, Kohli, & Maj, 2004; Goodwin & Franklin, 1994; Senge, Kleiner, Roberts, Ross, & Smith, 1994; Sterman, 1989). In The Beer Game all calculations are performed by hand on simple worksheets. This found favour:

"Doing the calculations by hand means we have to understand"

"The calculator half of the game hides details. Just give us a calculator and we can work it out"

Although the players' reception of the game was generally positive, clear written instructions are essential to make sure best use is made of the game session time. This comment was made by a player in the very first game session:

"Wasn't sure of what we were supposed to do"

Initially, instructions for playing the game were delivered by the researcher after the players had completed the pre-game survey and just before they started the game. For the second game session onwards, a one-page instruction sheet was emailed to each player a couple of days beforehand so they could be prepared.

The database of Simsoft game transactions showed that only three games had to be abandoned and restarted. It was observed that once teams had made the first couple of decisions, they were able to continue with too much trouble.

5. Discussion

The purpose of this research project was to see if and how games could contribute to better software project management education by helping software engineers and project managers explore some of the dynamic complexities of the field in a safe and inexpensive environment.

The major finding was that participants *were* learning as they played the game. However, the findings also suggested that games alone are not more effective than more traditional pedagogical means such as lectures, case studies, and readings. It also seems that simple games, and games in which the participants are able to relate game play to an external context, such as their real-world roles, are the most efficacious.

This section analyses and discusses the findings in more detail along the following broad analytic categories:

Games and learning in Simsoft.

Games in context.

The relative complexity of games.

5.1 Learning in Simsoft

The results showed that each group of participants (students, project managers, and software developers) improved their performance between the pre- and post-game tests. This suggests that the participants were constructing knowledge for themselves based on what they had experienced in the game. Comments from the participants supported this:

"Aha!"

"Now I see why"

When each group was further classified by years of experience in the field, the same improvement between the

pre- and post-game tests was seen, with the greatest improvement being in those with more experience. For example, students gained relatively less from the game than more experienced software developers and project managers.

Together these results suggest that learning *is* happening, but for some participants at least some level of *a priori* knowledge is necessary to make more sense of what is happening in the game. So, participants can learn some, but not all, of what they need to know from a game.

5.2 Games in Context

A common comment during the after-game gatherings, and something that was reflected in the post-game survey, was that most participants were prepared to invest greater time and effort in games such as Simsoft if the reward was deeper understanding of the problem domain.

With this in mind, one participant suggested running the game in real time, so that one week of real time equated to one week of project time. During the week, the team members could do research and discuss their options before coming to a carefully considered decision about their next step. This suggestion was influenced by a stock market game a number of participants had played the previous year. Players bought and sold shares on a fantasy stock exchange based on real prices published in the daily newspaper. The winner after three months was the team with the largest portfolio. In the week between submitting buy and sell orders, the players researched likely companies, scanned market reports, and took note of interest rate decisions, the price of oil and gold, and currency fluctuations to see how they might affect the market.

This suggestion represents a desire to put Simsoft more in context, by allowing the participants to step out of the fantasy world of the game, do some study, and then step back into the game with better knowledge. However, Simsoft, and all other software engineering management games discovered during a systematic search of the literature (Caulfield, Xia, Veal, & Maj, 2011), are played in one-off sessions. What players learned, must be learned within the hour or so of the game session. Of course, games can be replayed, but they must have sufficient depth to present alternate, engaging paths through the game in repeat. For even the most sophisticated game in the field, SimSE, players became bored when playing second and subsequent times (Navarro & van der Hoek, 2007).

One way to satisfy this desire for more depth, would be to play the game across multiple sessions over weeks or months as some participants have done with other games. In between, research could be undertaken in order to make the most informed decision.

For some participants, this break is necessary. Evaluation of the pre- and post-game scores showed that students gained relatively less from the game than more experienced project managers and developers. The following comment from a student is illustrative:

"I saw in the game aspects of theory covered at uni, but without knowing the theory first I probably wouldn't have recognised the significance."

That is, students in this research population didn't have the *a priori* knowledge needed to make full sense of the game's dynamics.

Playing the game over multiple, rather than single, sessions would more closely conform to the tenets of problem-based learning where participants begin their project with imperfect knowledge and then have to identify and learn what they needed in order to solve the issue at hand.

5.3 The Relative Complexity of Games

When asked, most Simsoft players said they preferred a board game to a fully computerized version because they could start playing more quickly without having to learn how to navigate a new user interface and without fear of making an unintended move. Apart from the mechanics of playing Simsoft, the simple design meant the state of the game and its underlying causal model were always visible.

The appeal of simplicity over complexity has been noted before. While complex games offer "the richest learning experience available, the game's very formidable appearance probably intimidated a number of players or forced them into a learning situation they were unprepared or unwilling to negotiate" (Wolfe, 1978, p.152). The next most effective game in Wolfe's study was found to be the least complex, supporting similar research that showed relatively simple games can provide essentially the same, if not more, benefits as the more complex (Butler, Pray, & Strang, 1979; Dempsey, et al., 2002; Meadows, 1999; Raia, 1966; Watt, 1977). Therefore, making games only as complex as necessary, or hiding unnecessary detail, could be a way of achieving the best learning outcomes while avoiding the player mortality (boredom and dropout) noted by Wolfe.

A board game also more easily fosters the collaboration needed in any team enterprise such as a software development project. When a computer or online game is played by multiple participants, likely at different physical locations, the basic cues of identity, personality, and body language are hidden. Without these cues, researchers have found that many computer games explicitly designed to be collaborative will degenerate into competitive games at worst or games in which “everyone just kind of does their own thing” (Zagal, Rick, & Hsi, 2006, p.25) at best.

In Simsoft, group play was viewed positively by most participants. It reflected real-world experience and also meant ideas and opinions could be shared:

“It was like [the agile] stand up meeting we have every morning”

“I thought software development was a solitary experience but it's not really”

“Everyone had a chance to offer an opinion”

Notwithstanding these positive aspects, any group activity may devolve into groupthink (Janis, 1971) in which the opinion of a dominant individual or clique prevails, possibly against reasonable evidence. In the Simsoft game sessions, no teams were larger than four participants and many participants were known to each, either professionally or socially, so there was ample opportunity to contribute to the discussions or even dispute the idea of a colleague or friend. There were also no more than four game sessions running at once, which meant the researcher was able to notice any participants standing back and gently prompt them for a contribution.

Few other software development game researchers have looked closely at these same aspects of game design. In (Hailey, Connelly, Stansfield, & Boyle, 2010), players were asked to rate game features such as graphics, realism of the characters, realism of the environment, and sound, but these were evaluations of the verisimilitude of these features, not their appropriateness to the task at hand. On this same rating of game features, collaboration ranked last or second last across all players, but this is to be expected in a single-player game. Similarly, other researchers (Baker, Oh Navarro, & van der Hoek, 2005; Navarro & van der Hoek, 2009; Zapata, 2010) asked their participants if they enjoyed playing the game or whether they found it engaging, but these questions ask the participants to evaluate a particular game's representation of its environment rather than its comparative complexity or its value as a collaborative tool.

5.4 Related Work

A recent systematic survey of games used in software engineering education (C. W. Caulfield, J. Xia, et al., 2011) found that, as a pedagogical device, they are becoming more common, particularly in Europe and the Americas, and students generally enjoyed playing them and felt they gained some value from the experience.

Simsoft differs from these other games in four main areas:

Simsoft is equally, if not more, concerned with *who* does the work in a software development as it is with process of *how* the work is done. This echoes the cover of Boehm's (1981) *Software Engineering Economics* which shows personnel is where the greatest productivity gains are possible.

Simsoft is largely a board game (with a small calculator component) in contrast to most other games that use a graphical user interface of varying levels of richness. Often the user interface is simply a conceit of the game for performing housekeeping functions and lends little to the real purpose. Other games that use playing cards or games boards contain an element of chance rather than skill.

Simsoft is cast at a level of detail at which the players can see the movement of individual pieces of work and individuals themselves. Games cast at higher levels can mask some fundamental project dynamics.

The research sample for this project is a mixture of students and experienced professionals rather than wholly students.

6. Conclusions

At the end of *War Games*, as Matthew Broderick and his girlfriend Ally Sheedy reflect on the world's narrow escape, the message is obvious: everyone has learned a lesson and blind reliance on computers is foolish. WOPR is quietly dismantled and won't be able to ask anyone else, “Shall we play a game?”

This paper posed the same question in a different context: shall we— should we— play games in software project management education? The answer, we believe, is a qualified, *yes*. The answer is qualified because our findings show that while games are useful pedagogical tools and are well-received by players, they are not sufficient in themselves and must be supplemented by other learning devices. Also, unless the games are designed with learning aforethought, they will probably miss their mark.

This project also points to some interesting directions for future research. Many participants said they preferred simple, collaborative games to complex games, and they were also prepared to play games in more depth if the reward was greater knowledge of the problem domain. The plan is to develop Simsoft further in these directions.

References

- Abt, C. C. (1970). *Serious Games*. New York: The Viking Press.
- Baker, A., Navarro, E., & Van der Hoek, A. (2005). An Experimental Card Game for Teaching Software Engineering Processes. *The Journal of Systems and Software*, 75(1 – 2). <http://dx.doi.org/10.1016/j.jss.2004.02.033>
- Barell, J. (2006). *Problem-Based Learning: An Inquiry Approach* (2nd edition ed.). Thousand Oaks: Corwin Press.
- Barrows, H. S., & Tamblyn, R. (1976). An Evaluation of Problem-Based Learning in Small Groups Utilizing a Simulated Patient. *Journal of Medical Education*, 51(1), 52 – 54. <http://www.eric.ed.gov/ERICWebPortal/detail?accno=EJ131167>
- Bloomberg, L., Dale, & Volpe, M. (2008). *Completing Your Qualitative Dissertation*. Thousand Oaks: Sage Publications.
- Boehm, B., Egyed, A., Port, D., Shah, A., Kwan, J., & Madachy, R. (1998). A Stakeholder Win–Win Approach to Software Engineering Education. *Annals of Software Engineering*, 6(1), 295 - 321. <http://dx.doi.org/10.1023/A:1018988827405>
- Boehm, B. W. (1981). *Software Engineering Economics*. Sydney: Prentice-Hall.
- Bourque, P., Dupuis, R., Abran, A., Moore, J. W., & Tripp, L. (1999). The Guide to the Software Engineering Body of Knowledge. *IEEE Software*, 16(6), 35 - 44. <http://dx.doi.org/10.1109/52.805471>
- Brereton, O. P., Lees, S., Bedson, R., Boldyreff, C., Drummond, S., Layzell, P. J., et al. (2000). Student Group Work Across Universities: A Case Study in Software Engineering. *IEEE Transactions on Education*, 43(4), 394 – 399. <http://dx.doi.org/10.1109/13.883348>
- Brooks, F. P. (1987). No Silver Bullet: Essence and Accidents of Software Engineering. *IEEE Computer*, 20(4), 10 – 19. <http://doi.ieeeecomputersociety.org/10.1109/MC.1987.1663532>
- Butler, R. J., Pray, T. F., & Strang, D. R. (1979). An Extension of Wolfe's Study of Simulation Game Complexity. *Decision Sciences*, 10, 480 – 486. <http://dx.doi.org/10.1111/j.1540-5915.1979.tb00038.x>
- Caillois, R. (1961). *Man, Play and Games* (M. Barash, Trans.). New York: Free Press of Glencoe.
- Caulfield, C. W., Veal, D., & Maj, S. P. (2011a). Implementing System Dynamics Models in Java. *International Journal of Computer Science and Network Security*, 11(7), 43 – 49.
- Caulfield, C.W., Veal, D., & Maj, S. P. (2011b). Teaching Software Engineering Project Management – A Novel Approach for Software Engineering Programs. *Modern Applied Science*, 5(5), 87 – 104. <http://dx.doi.org/10.5539/mas.v5n5p87>
- Caulfield, C. W., Kohli, G., & Maj, S. P. (2004). Sociology in Software Engineering. *Proceedings of Proceedings of the 2004 American Society for Engineering Education Annual Conference & Exposition*, Salt Lake City.
- Caulfield, C. W., Xia, J., Veal, D., & Maj, S. P. (2011). A Systematic Survey of Games Used for Software Engineering Education. *Modern Applied Science*, in press.
- Cheng, Y. P., & Lin, J. M. C. (2010). A Constrained and Guided Approach for Managing Software Engineering Course Projects. *IEEE Transactions on Education*, 53(3), 430 – 436. <http://dx.doi.org/10.1109/TE.2009.2026738>
- Creswell, J. W. (2009). *Research Design: Qualitative, Quantitative, and Mixed Methods Approaches* (3rd edition ed.). Thousand Oaks: Sage Publications.
- DeMarco, T. (1982). *Controlling Software Projects*. New York: Yourdon Press.
- DeMarco, T., & Lister, T. (1999). *Peopleware: Productive Projects and Teams* (2nd edition ed.). New York: Dorset House Publishing Co.
- Dempsey, J. V., Haynes, L. L., Lucassen, B. A., & Casey, M. S. (2002). Forty Simple Computer Games and What They Could Mean to Educators. *Simulation & Gaming*, 33(2), 157 – 168. <http://dx.doi.org/10.1177/1046878102332003>

- Eldredge, D. L., & Watson, H. J. (1996). An Ongoing Study of the Practice of Simulation in Industry. *Simulation & Gaming*, 27(3), 375 – 386. <http://dx.doi.org/10.1177/1046878196273008>
- Faria, A. J. (1987). A Survey of the Use of Business Games in Academia and Business. *Simulation & Games*, 18(2), 207 – 224. <http://dx.doi.org/10.1177/104687818701800204>
- Faria, A. J. (1998). Business Simulation Games: Current Usage Levels—An Update. *Simulation & Gaming*, 29(3), 295 – 308. <http://dx.doi.org/10.1177/1046878198293002>
- Faria, A. J., & Wellington, W. J. (2004). A Survey of Simulation Game Users, Former-Users, and Never-Users. *Simulation & Gaming*, 35(2), 178 – 207. <http://dx.doi.org/10.1177/1046878104263543>
- Faria, A. J., & Wellington, W. J. (2005). Validating Business Gaming: Business Game Conformity with PIMS Findings. *Simulation & Gaming*, 36(2), 259 – 273. <http://dx.doi.org/10.1177/1046878105275454>
- Gee, J. P. (2007). *Good Video Games and Good Learning: Collected Essays on Video Games, Learning and Literacy*. New York: Peter Lang Publishing.
- Goodwin, J. S., & Franklin, S. G. (1994). The Beer Distribution Game: Using Simulation to Teach Systems Thinking. *Journal of Management Development*, 13(8), 7 – 15. <http://dx.doi.org/10.1108/02621719410071937>
- Gotterbarn, D. (1999). How the New Software Engineering Code of Ethics Affects You. *IEEE Software*, 16(6), 58 – 64. <http://doi.ieeecomputersociety.org/10.1109/52.805474>
- Guba, E. G., & Lincoln, Y. S. (2005). Paradigmatic Controversies, Contradictions, and Emerging Confluences. In N. K. Denzin & Y. S. Lincoln (Eds.), *The Sage Handbook of Qualitative Research* (3rd edition ed., pp.191 – 215). Thousand Oaks: Sage Publications.
- Hainey, T., Connelly, T. J., Stansfield, M., & Boyle, E. A. (2010). Evaluation of a Game to Teach Requirements Collection and Analysis in Software Engineering at Tertiary Education Level. *Computers & Education*, 56(1), 21 – 35. <http://dx.doi.org/10.1016/j.compedu.2010.09.008>
- Heldman, K. (2007). *PMP: Project Management Professional Exam Study Guide* (4th edition ed.). San Francisco: Sybex.
- iSSEc Project. (2009). *Graduate Software Engineering 2009 (GSWE2009): Curriculum Guideline for Graduate Degree Programs in Software Engineering*.
- Janis, I. L. (1971). Groupthink. *Psychology Today*, 5(5), 43 - 46, 74 – 76.
- Johns-Boast, L., & Patch, G. (2010). A Win-Win Situation: Benefits of Industry-Based Group Projects. *Proceedings of Australasian Association for Engineering Education Conference (AaeE 2010)*.
- Joint IS2010 Curriculum Task Force. (2010). *Curriculum Guideline for Undergraduate Degree Programs in Information Systems*: Association for Computing Machinery and Association for Information Systems.
- Joint Task Force on Computing Curriculum. (2004). *Software Engineering 2004: Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering*: IEEE Computer Society/Association for Computing Machinery.
- Kirk, J., & Miller, M. L. (1986). *Reliability and Validity in Qualitative Research*. London: Sage Publications.
- Kruchten, P. (2005). Editor's Introduction: Software Design in a Postmodern Era. *IEEE Software*, 22(2), 16-18. <http://doi.ieeecomputersociety.org/10.1109/MS.2005.38>
- Kruchten, P. B. (2004). The Nature of Software: What's So Special About Software Engineering? [Online] Available: www.ibm.com/developerworks/rational/library/4700.html
- Lincoln, Y. S., & Guba, E. G. (1984). *Naturalistic Inquiry*. London: Sage Publications.
- Marshall, M. N. (1996). Sampling for Qualitative Research. *Family Practice*, 13(6), 522 – 525.
- Maxwell, J. A. (2004). *Qualitative Research Design: An Interactive Approach* (2nd edition ed.). Thousand Oaks: Sage Publications.
- Maxwell, N. L., Mergendoller, J. R., & Bellisimo, Y. (2004). Developing a Problem-Based Learning Simulation. *Simulation & Gaming*, 35(4), 488 - 498. <http://dx.doi.org/10.1177/1046878104264789>
- McCall, J. (2011). *Gaming the Past: Using Video Games to Teach Secondary History* London: Routledge.
- McConnell, S. (2004). *Professional Software Development*. Boston: Addison-Wesley.
- McKenna, R. J. (1991). Business Computerized Simulation: The Australian Experience. *Simulation & Gaming*,

22(1), 36 – 62. <http://dx.doi.org/10.1177/1046878191221003>

Meadows, D. L. (1999). Learning to Be Simple: My Odyssey with Games. *Simulation & Gaming*, 30(3), 342 – 351. <http://dx.doi.org/10.1177/104687819903000310>

Michael, D., & Chen, S. (2005). *Serious Games: Games That Educate, Train, and Inform*. Boston: Thomson Course Technology PTR.

Miller, J. G. (1978). *Living Systems*. New York: McGraw-Hill Book Company.

Navarro, E. O., & van der Hoek, A. (2007). Comprehensive Evaluation of an Educational Software Engineering Simulation Environment. *Proceedings of The Twentieth Conference on Software Engineering Education and Training*, July 2007 Dublin, Ireland

Navarro, E. O., & van der Hoek, A. (2009). Multi-Site Evaluation of SimSE. *Proceedings of The 40th ACM Technical Symposium on Computer Science Education* March 3 – 7 Chattanooga, Tennessee.

Naveda, J. F., & Seidman, S. B. (Eds.). (2006). *IEEE Computer Society Real-World Software Engineering Problems: A Self-Study Guide for Today's Software Professional*. Hoboken: John Wiley & Sons.

Osterweil, L. (1987). Software Processes are Software Too. *Proceedings of Proceedings of the 9th International Conference on Software Engineering* Monterey, California.

Patton, M. Q. (2002). *Qualitative Research and Evaluation Methods* (3rd edition ed.). Thousand Oaks: Sage Publications.

Perla, P. P. (1990). *The Art of Wargaming: A Guide for Professionals and Hobbyists*. Annapolis, Maryland: Naval Institute Press.

Polack-Wahl, J. A. (2006). Lessons Learned From Different Types of Projects in Software Engineering. *Proceedings of International Conference on Frontiers in Education: Computer Science & Computer Engineering*, June 26-29, 2006, Las Vegas, Nevada

Prensky, M. (2006). *Don't Bother Me Mom – I'm Learning!*. St. Paul, Minnesota: Paragon House Publishers.

Raia, A. P. (1966). A Study of the Educational Value of Management Games. *The Journal of Business*, 39(3), 339 – 352. <http://dx.doi.org/doi:10.1086/294863>

Richards, L. (2009). *Handling Qualitative Data* (2nd edition ed.). Thousand Oaks: Sage Publications.

Riddell, R. (1997, April). Doom Goes to War. *Wired*, 5, 113 – 118, 164 – 166.

Robillard, P. N., & Robillard, M. (1998). Improving Academic Software Engineering Projects: A Comparative Study of Academic and Industry Projects. *Annals of Software Engineering*, 6(1), 343 - 363. <http://dx.doi.org/10.1023/A:1018925902814>

Rossman, G. B., & Rallis, S. F. (1998). *Learning in the Field: An Introduction to Qualitative Research*. Thousand Oaks: Sage Publications.

Salen, K., & Zimmerman, E. (Eds.). (2005). *The Game Design Reader: A Rules of Play Anthology* Cambridge, Massachusetts: The MIT Press.

Savin-Baden, M. (2003). *Facilitating Problem-Based Learning*. Maidenhead: The Society for Research into Higher Learning & Open University Press.

Savin-Baden, M., & Major, C. H. (2004). *Foundations of Problem-Based Learning*. Maidenhead: The Society for Research into Higher Learning & Open University Press.

Schrage, M., & Peters, T. (1999). *Serious Play : How the World's Best Companies Simulate to Innovate*: Harvard Business School Press.

Senge, P. M., Kleiner, A., Roberts, C., Ross, R. B., & Smith, B. J. (1994). *The Fifth Discipline Fieldbook*. London: Nicholas Brealey Publishing.

Sterman, J. D. (1989). Modeling Managerial Behavior: Misperceptions of Feedback in a Dynamic Decision Making Environment. *Management Science*, 35(3), 321 – 339. <http://dx.doi.org/10.1287/mnsc.35.3.321>

Strauss, A. L. (1987). *Qualitative Analysis for Social Scientists*. Cambridge: Cambridge University Press.

Suits, B. (1967). What is a Game?. *Philosophy of Science*, 34(2), 148 – 156. <http://www.jstor.org/stable/186102>

Tang, C., Lai, P., Tang, W., Davis, H., Frankland, S., Oldfield, K., et al. (1997). Developing a Context-Based PBL Model. In J. Conway, R. Fisher, L. Sheridan-Burns & G. Ryan (Eds.), *Research and Development in*

Problem Based Learning: Integrity, Innovation, Integration (pp.588 – 589). Newcastle: Australian Problem Based Learning Network.

Watt, K. E. F. (1977). Why Won't Anyone Believe Us?. *Simulation*, 28(1), 1 – 3. <http://dx.doi.org/10.1177/003754977702800102>

Wolfe, J. (1978). The Effects of Game Complexity on the Acquisition of Business Policy Knowledge. *Decision Sciences*, 9(1), 143 – 155. <http://dx.doi.org/10.1111/j.1540-5915.1978.tb01373.x>

Zagal, J. P., Rick, J., & Hsi, I. (2006). Collaborative Games: Lessons Learned from Board Games. *Simulation & Gaming*, 37(1), 24 — 40. <http://dx.doi.org/10.1177/1046878105282279>

Zapata, C. M. (2010). A Classroom Game for Teaching Management of Software Companies. *Dyna*, 77(163), 290 – 299.

Zyda, M. (2007). Creating a Science of Games. *Communications of the ACM*, 50(7), 26 – 29. <http://dx.doi.org/10.1145/1272516.1272535>

Table 1. Comparison of players pre- and post-game test scores

Role and Experience (in years)	n	Average pre-test score (out of 8)	Average pre-test score (out of 8)	Difference Between Pre- and Post-Game Scores
Students	17	4.64 (SD = 0.861)	5.41 (SD = 1.460)	+0.77
0 to 1 years	17	4.64 (SD = 0.861)	5.41 (SD = 1.460)	+0.77
Software Developers	30	5.53 (SD = 0.995)	6.33 (SD = 1.107)	+0.80
0 to 1	0			
2 to 5 years	14	5.57 (SD = 1.089)	6.07 (SD = 1.268)	+0.50
5 to 10 years	11	5.72 (SD = 1.009)	6.818 (SD = .0750)	+1.098
10 to 15 years	5	5.00 (SD = 0.707)	6.00 (SD = 1.224)	+1.00
15+ years	0			
Project Managers	12	4.66 (SD = 1.497)	5.42 (SD = 2.020)	+0.76
0 to 1	0			
2 to 5 years	6	4.5 (SD = 2.073)	5.00 (SD = 2.529)	+0.50
5 to 10 years	1	5.00 (SD = NA)	6.00 (SD = NA)	+1.00
10 to 15 years	4	4.75 (SD = 0.957)	5.75 (SD = 1.892)	+1.00
15+ years	1	5.00 (SD = NA)	6.00(SD = NA)	+1.00
	59	5.10 (SD = 1.155)	5.88 (SD = 1.486)	+0.78

Table 2. Participants' responses when asked whether they thought Simsoft was easy or difficult to play

Role and Experience (in years)	n	Average Response
Students	17	3.17 (SD = 0.528)
0 to 1 years	17	3.17 (SD = 0.528)
Software Developers	30	2.93 (SD = 0.253)
0 to 1	0	
2 to 5 years	14	2.92 (SD = 0.267)
5 to 10 years	11	3.00 (SD = 0.000)
10 to 15 years	5	2.80 (SD = 0.447)
More than 15 years	0	
Project Managers	12	2.58 (SD = 0.514)
0 to 1	0	
2 to 5 years	6	2.83 (SD = 0.408)
5 to 10 years	1	3.00 (SD = NA)
10 to 15 years	4	2.25 (SD = 0.500)
More than 15 years	1	2.00 (SD = NA)
	59	2.93 (SD = 0.449)

Table 3. Players' evaluation of game features

Feature	Average (1 = very bad, 5 = very good; or 1 = strongly disagree, 5 = strongly agree)
Written instructions	Average = 4.44, SD = 0.771
The game was interesting	Average = 4.37, SD = 0.963
Realistic scenario	Average = 4.37, SD = 0.692
Navigation around the game	Average = 4.22, SD = 0.744
Game logic was apparent	Average = 4.18, SD = 0.730
Useful to work in teams	Average = 4.15, SD = 0.714
Prefer game-board version	Average = 3.98, SD = 0.754