1-1-2011

Teaching software engineering project management-A novel approach for software engineering programs

Craig Caulfield
Edith Cowan University

David Veal
Edith Cowan University

Stanislaw Maj
Edith Cowan University

10.5539/mas.v5n5p87

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 GWwE2009 must be mastered at level 2 or higher. Moreover, many more topics in GWwE2009 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 is 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 Peopleware*. 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). *Peopleware: 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 IEEIII 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	
1. Co-design: good learning means that players feel they are active agents (producers) not just passive recipients (consumers). 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.	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.
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.	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

 ISSN 1913-1844 E-ISSN 1913-1852

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.	players to debate their decisions.
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. 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.	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: "I wish I have [sic] this power at work"
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." 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.	The players had full control over their workforce, subject to budget constraints and hiring delays.
II. Problem Solving	
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.	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.
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.	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

Players also need feedback so even if they fail, they have an idea of what must be done next time.	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.
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.	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.
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).	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.
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.	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
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. 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	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.

 ISSN 1913-1844 E-ISSN 1913-1852

later in the game.	
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. 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.	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.
III. Understanding	
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. 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.	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: "I see my part in the machinery now"
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.	For experienced software developers and project managers, thinking about their work in concrete rather than abstract terms is easy and connections can be made: "Now I see why" "I hope that future versions will let me set up specific scenario and play them out. That would really help me in my work" 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: "I must remember this"

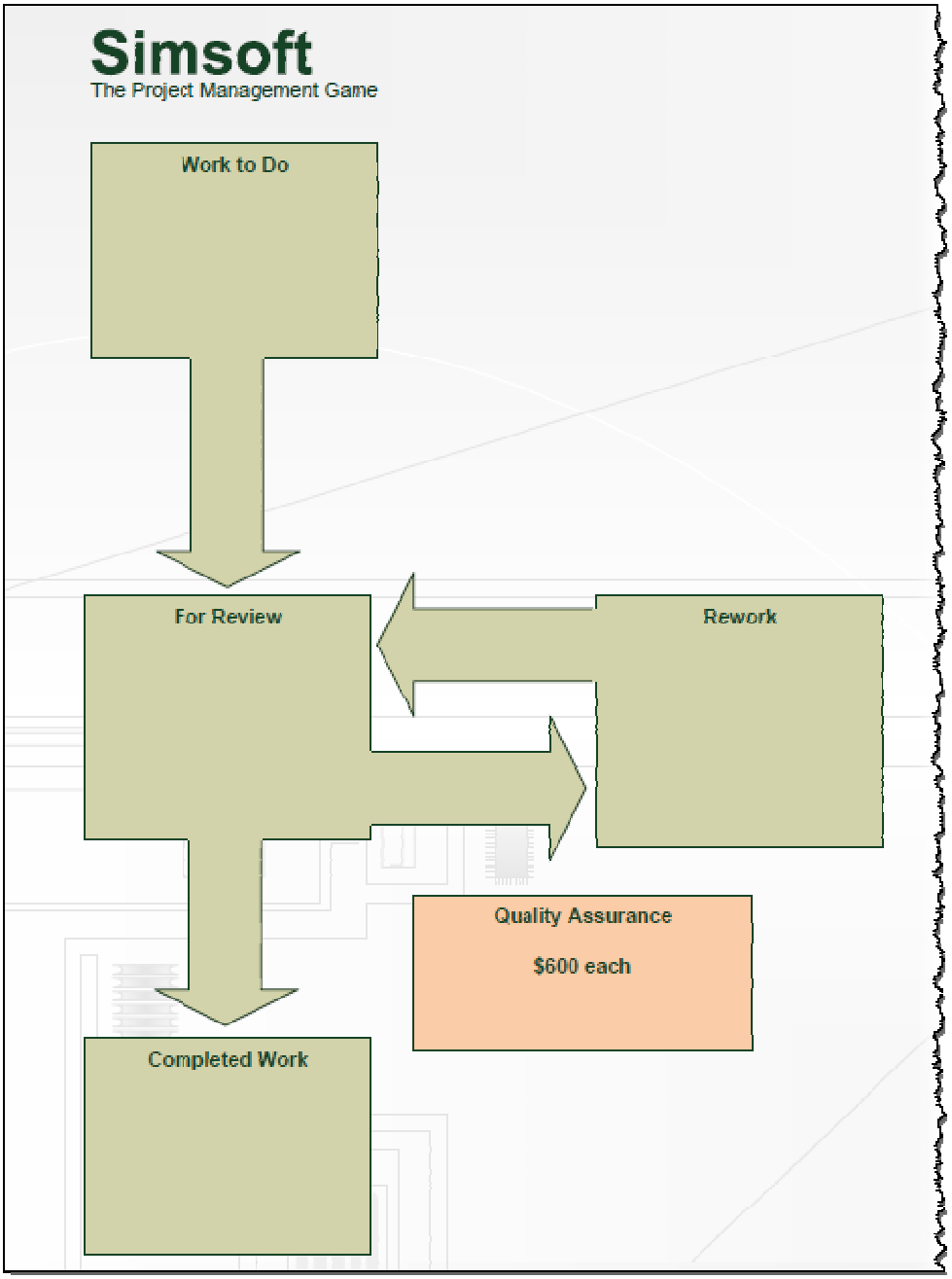

Figure 1. Left-hand side of the Simsoft game board showing the work to be done

ISSN 1913-1844 E-ISSN 1913-1852

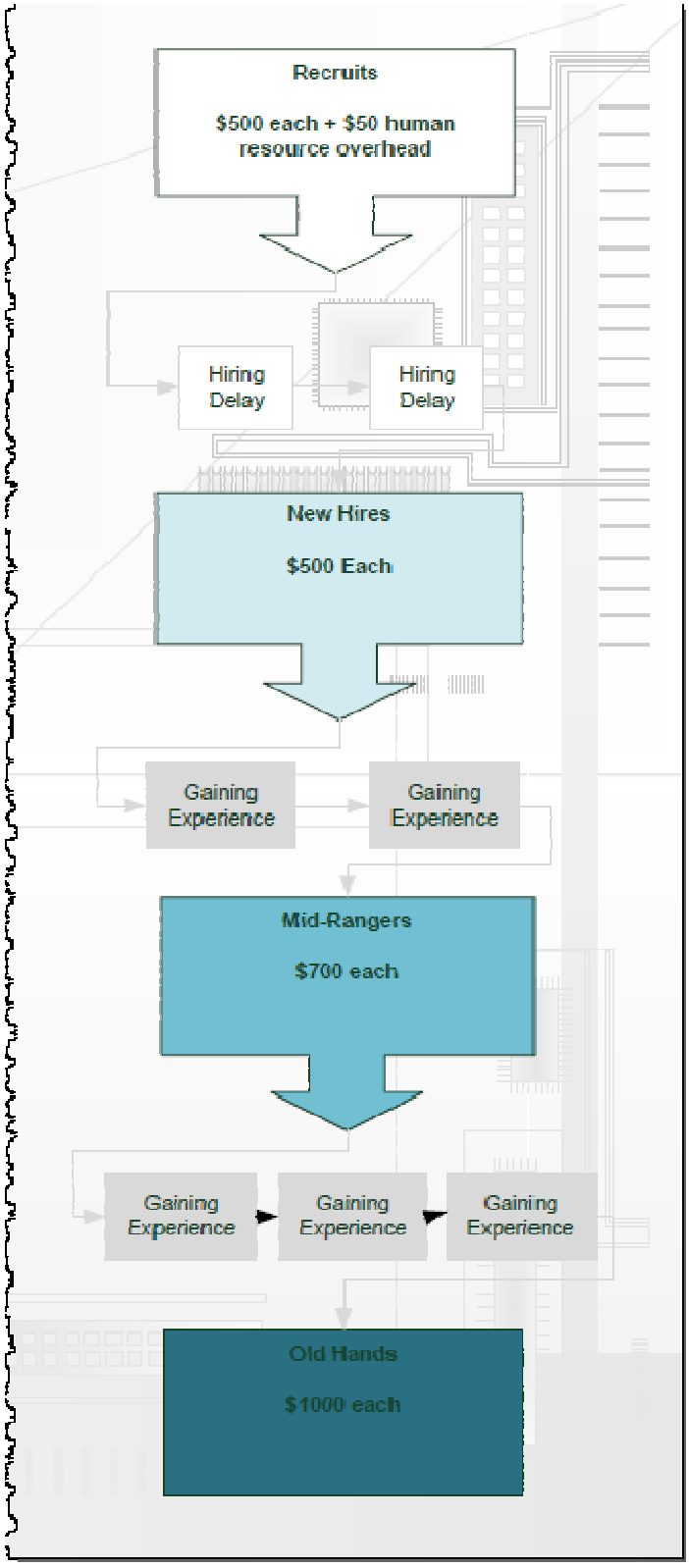

Figure 2. Right-hand side of the Simsoft game board showing the human resources of the project

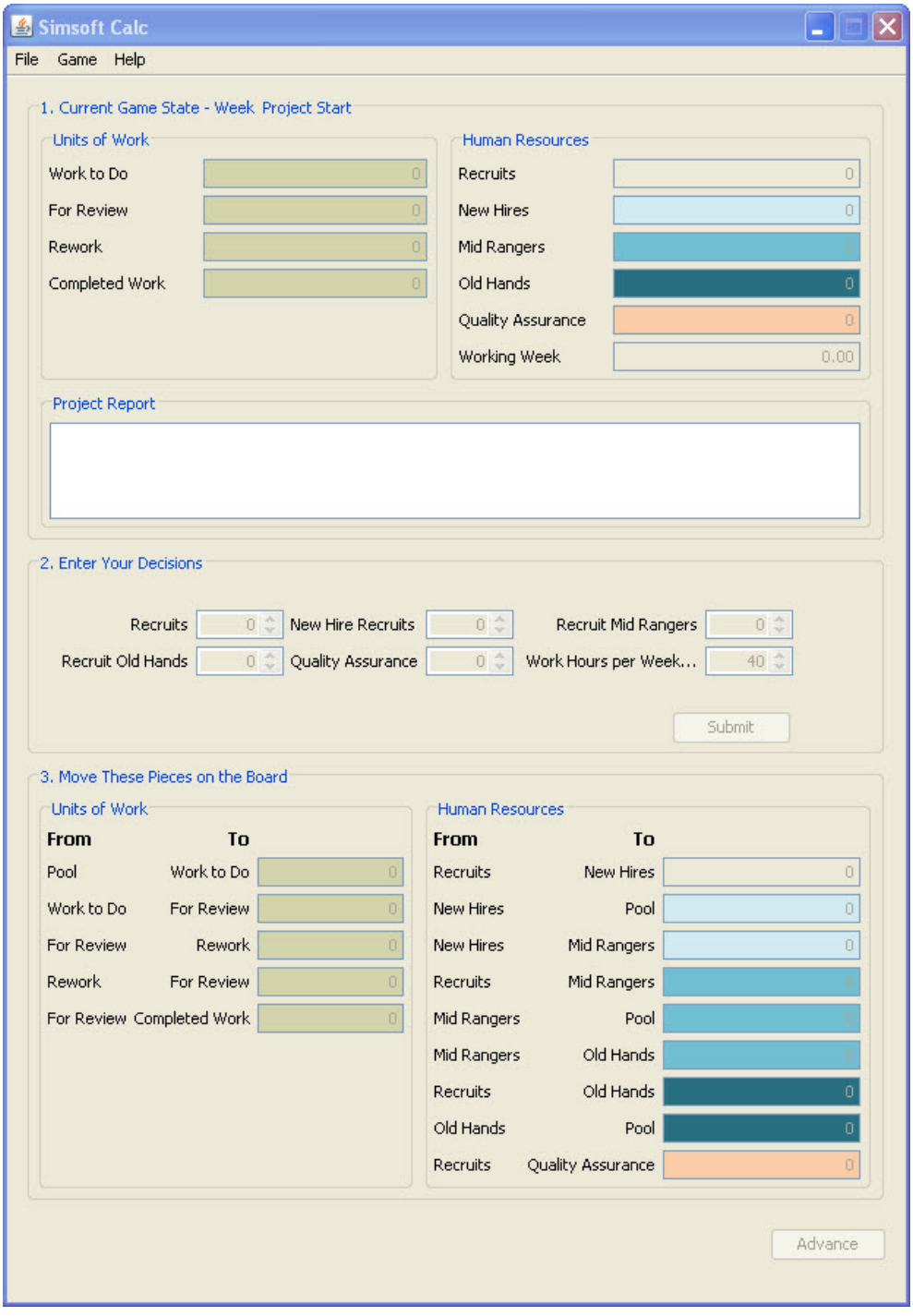

Figure 3. Simsoft dashboard

ISSN 1913-1844 E-ISSN 1913-1852