Adapting Game Technology to Support Individual and Organizational Learning

Emily Oh and André van der Hoek
Institute for Software Research
University of California, Irvine
Irvine, CA 92612–3425 USA
emilyo@uci.edu and andre@ics.uci.edu

Abstract
It is well known that traditional educational techniques can be complemented by simulation to achieve a more effective learning experience. One would expect the same phenomenon to be true in software development. However, the simulation techniques used thus far have not been effective. This paper introduces a novel approach to simulation for software development education that is based on the adaptation of game technology. Specifically, we propose to build a simulation environment that interacts with its users much like games such as SimCity and The Sims. In providing direct, graphical feedback, we hypothesize that this approach allows individuals to develop an understanding of the software processes used in their organization, while organizations as a whole benefit from the ability to explore different approaches to their software development process.

1. Introduction

The ever-increasing presence of software in our society necessarily demands faster, higher-quality software development processes that do not impose a large financial burden on the organizations employing them. Consequently, a great need has arisen to find effective ways to improve the software process by building up knowledge both on the organizational and individual levels. In response to this need, much research has been devoted to software development education and training in recent years—not only within organizations themselves, but also at the universities and colleges that are normally first in introducing software development practices and processes to students.

Typically, this kind of education and training involves one or more of the following:

1. A series of theoretical lectures in which the nature of software development is introduced, covering life cycle models, methodologies, modeling techniques, example processes, and desired properties of the set of artifacts to be produced.

2. A small project in which software development is practiced, some example processes are followed, and direct feedback is provided by the project teacher.

3. A highly focused, "shotgun"-like seminar in which a particular aspect of software development is meticulously described, analyzed, and applied to the situation at hand.

However, none of these approaches provide sufficient theoretical background and sufficient practical experience to adequately teach the software process in its fullest. Particularly in the context of individual and organizational learning, this presents a problem: individuals need to understand their role within the larger process and organizations need to understand the process as a whole in order to be able to make any improvements.

This paper describes our initial approach to building an educational software development simulation environment that facilitates individual and organizational learning. The environment supports individual learning by allowing a user to operate in a simulated environment that mimics the particular process of their organization. Organizational learning is promoted by allowing an organization itself to examine the simulated process and anticipate, through experimentation, the effects, benefits, and drawbacks of different decisions.

The remainder of this paper is organized as follows. Section 2 discusses the educational challenges surrounding the inherent nature of software development. Section 3 provides background on educational simulation techniques, emphasizing those used in software development. Section 4 discusses simulation games and their relevance to this research. Section 5 presents the central hypothesis underlying this project. Section 6 describes our initial approach to developing an educational software development simulation environment. Our conclusions and plans for future work are presented in Section 7.

2. Nature of software development

Software development has a number of characteristics that make teaching the software process with traditional mechanisms (such as lectures or class projects) rather difficult. Specifically, the combined difficulties posed by the following five characteristics have sparked our research into new methods of teaching the software development process.

1. **Software development is non-linear.** Activities, tasks, and phases are repeated. Multiple events happen at the same time. Managing a project in the same way every time will generally not produce the same outcome due to the presence of several random factors (e.g., human factors, new technical advances, customer's erratic behaviors). Although this kind of lesson can be conveyed in a lecture, practical experience is necessary to really understand the powerful nature of the non-linearity of software development. Unfortunately, the time schedule associated with most class projects does generally not allow one to repeat the same process over and over.

2. **Software development involves several intermediate steps and continuous choices between multiple, viable alternatives.** Even with careful planning, all possible events that can occur cannot be anticipated at the start of a project. Difficult decisions must be made, tradeoffs must be considered, and conflicts must be handled constantly throughout the software life cycle. Generally a project moves along following precisely the steps prescribed by the instructor. Other than deciding upon the actual content of a design and implementation, decisions regarding the process do not have to be made. This is precisely the problem—process decisions are often some of the hardest ones to make.

3. **Software development may exhibit dramatic effects with non-obvious causes.** Although the software process has several more obvious cause-and-effect relationships (e.g., changes made in later phases of development are more time-consuming and expensive than those made early in development), there are several common situations that may arise in which the cause is not so apparent. For example, it is well known that adding people to a project that is already late typically makes that project later (Brooks' Law) [6]. However, the underlying cause of this phenomenon, namely the increased amount of communication necessitated by the greater demands for learning and coordination among a larger number of people, may not necessarily be so clear. Once again, some of these lessons can be taught in a lecture, but practical experience is required to fully understand their implications. Most class projects however, are of limited size and scope and—other than an influence on a students' grade—have no real consequences.

4. **Software engineering involves multiple stakeholders.** People other than developers and project managers, including customers and persons in non-development roles in an organization, all make decisions that affect development. In a typical class project, these roles are ignored. At best, an instructor plays the role of a customer and puts some hypothetical restrictions on the process on behalf of a virtual management team. It is desired, however, that students be able to learn the influence of all potential stakeholders and the powers that their desires exhibit over the software development process.

5. **Software engineering often has multiple, conflicting goals.** The software process is fraught with tradeoffs between such things as quality versus cost, timeliness versus thoroughness, or reliability versus performance. In a typical software development class, these tradeoffs are introduced in lectures, but normally ignored in the class project. A relatively complete requirements document, a working design, and a prototype implementation are usually sufficient for a passing grade. This is a significant simplification and hardly touches upon the difficult tradeoffs that are often present in practice.

It is crucial for any educational experience of the software development process to communicate these issues in order to create a full understanding of the depth and complicated nature of the software process. Although simple illustrations of these kinds of complications are nowadays becoming more standard in software development education [9], none of the individual and organizational software process learning techniques developed thus far have achieved a high level of inclusiveness for all of the above dimensions. Most problems are introduced in lectures, but the practical experience falls far short of what a student can expect "in the real world", thus preparing them for a rather significant shock once they move to industry and requiring industry to provide their own, comprehensive training programs.

3. Simulation

Simulation is a powerful educational tool that has been used successfully in many different settings, such as flight simulation [22], military training [16], and hardware design [7]. Learning via simulation provides significant educational benefits: valuable experience is accumulated without the potentially dramatic consequences that may occur in case of failure. Moreover, unknown situations can be introduced and practiced, experiences can be repeated, alternatives can be explored, and a general freedom of experimentation and "play" is promoted in the training exercise [15]. We believe simulation is the ideal platform upon which to teach software development and, in particular, software development processes. As compared to lectures and employee training seminars, simulation has the distinct benefit of showing and teaching users cause and effect in a practical manner: if they make a wrong decision in the simulation, it will (hopefully) become clear to them because the simulation environment will show them certain undesired effects. As compared to on-the-job training, simulation has the distinct benefit of being much quicker: one does not have to wait days, weeks, or even months to see the effects of a decision, since the simulation environment is able to operate at a faster pace than real life. In general, simulation allows a practical experience without the additional, distracting burden of having to produce project deliverables.

In essence, education via simulation allows students to learn cause and effect relationships and consequences of decisions in a practical manner, while significantly reducing costs in time, money, and adverse real-world consequences. Moreover, simulation promotes practice with unknown situations, allows repeat experiences and exploration of different alternatives, provides the freedom of experimentation and "play" that is typically absent in other teaching techniques, and promotes insight into the relationships among the various components of the process, as well as an overall understanding of the behavior of the process being modeled. Through all of this, simulation equips students with an increased ability to predictably understand the behavior of real-world systems [15].

Several software process simulators have already been developed [4, 14, 21]. All of these operate according to the following basic philosophy: create a model of the real world, choose a set of input parameters, run the model, and examine the outputs together with traces of the model simulation to understand the workings of the environment and discover possible areas for improvement. This approach has been used for both organizational learning [19, 23] (providing management of an organization with insights into the process as a whole and the influences on the process of such decisions as to hire more personnel or to introduce specific software tools and methodologies) and individual learning [8, 10] (supporting individuals in their day-to-day decisions by providing them with learning tools and simulations to predict the outcome of some of their tasks based on certain input parameters). However, these simulations are, for the most part, continuous (they run without interruption) and non-graphical (at best, consisting of buttons, graphs, tables, diagrams, and text). This leaves the user with a passive role and portrays software development as a linear, non-interactive, one-sided process. Moreover, none of the simulators have achieved an appropriate level of interactivity, collaboration, and completeness while competently addressing each of the five aforementioned fundamental aspects of software development.

4. Games

Simulation games represent a tremendous source of experience that can be leveraged in creating new software development training and education approaches. A class of games that is particularly relevant is the one derived from the so-called "adventure games" of the olden days—now represented by such popular games as SimCity [11], The Sims [12], Escape from Monkey Island [17], Myst [20], Ultima Online [13], various multi-user dungeon games (MUDs) [18, 24, 26], MUDs-object-oriented (MOOs) [1-3] and many others. In these games, players live their lives in a virtual world and have to work towards achieving certain, sometimes conflicting, goals. For example, in The Sims the goal is to live as prosperous a life as possible by purchasing a house, working to earn money, falling in love, buying food and furniture, and performing other domestic kinds of activities. However, The Sims is full of complicated tradeoffs that hinder the achievement of all of the goals in parallel. For example, purchasing a bigger home (necessary to house a family) leads to fewer funds available for purchasing food (necessary to stay healthy). These kinds of tradeoffs are inherent to The Sims (and other games) and form the essence of its game play: the eventual outcome is determined by player's choices to work towards certain goals while ignoring others.

An important characteristic of these games is that the game itself is typically responsible for simulating random events and providing all auxiliary characters, along with their own, often unexpected, behaviors. The player must handle and interact with these characters and events in order to successfully advance in the game.

It is interesting to observe that these games exhibit strengths in addressing exactly those dimensions that make teaching software development processes so difficult:

- **They are non-linear.** Multiple events happen at the same time; one has to frequently interrupt certain activities to tend to others; and generally playing the game in the same way every time will not lead to the same results, due to the presence of several random factors in the simulated characters and events.

- **They allow for the exploration of alternatives.** All games allow a player to save the state of the game, in effect providing a checkpoint ability that can be leveraged to explore different directions without committing oneself—simply returning to the saved state allows for exploration of a different alternative.

- **They exhibit dramatic consequences and illustrate their causes.** For example, if a player in the Sims has a guest in their house, and after a while the guest leaves with an angry facial expression, the game will tell the player why they left (e.g., the player did not feed them, the player did not entertain them, the player did not talk to them enough). Although not real, the graphical illustration of some of the possible consequences in these games (which range from the player actually being killed, to buildings being destroyed by natural disasters, to dirty houses being invaded by rats) has a profound impact on the player and their advancement throughout the game.

- **They generally involve multiple stakeholders.** In some games, these stakeholders are represented by the different players that each try to optimize their own results. In other, single-user games, the game simulation provides the stakeholders. For example, SimCity has unions and green party representatives that the player has to keep happy in making decisions regarding city planning.

- **They involve multiple, conflicting goals.** As exemplified above with the house purchase example, the games involve optimizing multiple goals that sometimes interfere with each other. Player's actions inherently weigh certain goals as more important than others, and generally lead to the attainment of some goals and only the partial fulfillment of others.

On top of that, these games illustrate many other examples of good and effective design that can be leveraged in creating a simulation environment for software development. They are fun to play, encourage experimentation, usually have an excellent graphical user interface, have immediate as well as time-delayed cause and effect relations, and bring the player into unexpected, unknown situations that need to be resolved. It is clear that a careful study of how games achieve all of these properties is essential for us to be able to build a comparable kind of simulation environment for software development education.

5. Hypothesis

We observe that simulation game technology is an excellent vehicle for reducing the difficulties faced in teaching the software development process. It should come as no surprise, therefore, that the central hypothesis underlying our research is the belief that the creation of a game-like, educational software development simulation environment is the answer to solving the problem of adequately teaching the software development process. Using such an environment, the software development process can be practiced in a rapid, real-world-like setting without the additional burden of actually having to produce artifacts. Of course, we do not propose that these simulation environments replace existing techniques such as lectures and class projects. Instead, we believe that simulation best serves in a complementary fashion—used in parallel with existing techniques to teach a broad perspective of software development. In particular, lectures are still required to introduce the topics to be simulated and class projects are still required to demonstrate and reinforce some of the lessons learned in the lectures and simulations.

We furthermore observe that an educational software engineering simulation environment is as useful for individual learning as it is for organizational learning. While "playing" in such an environment, individuals can learn about their place in the software development process, understand the impact of their decisions (and non-decisions!), and generally get an overview of the broad software development process as it exists in an organization. Organizations can use the simulation environment to experiment with ways in which to select and instrument their organizational process. Specifically, by setting up the environment with different processes and running a series of simulations, they can understand the tradeoffs of different organizational structures and processes, see visual feedback on the consequences of each choice, and choose the particular process to be instituted in their organization. Of course, most often this process will be incremental: organizations can periodically use the simulator to fine-tune their process.

6. Approach

We are in the very preliminary stages of addressing the above hypothesis. Our research thus far has concentrated on setting requirements and creating an initial design for our version of an educational software development simulation environment. The most important lesson learned in this effort is the fact that we need a three-pronged approach. Specifically, we will need to collect the fundamental rules of software development, design and implement the environment, and create models that encapsulate sets of rules and are used to drive the simulation environment with particular scenarios.

6.1. Fundamental Rules of Software Development

Like any other discipline, software development has many underlying empirical rules. For example, it is well known that skipping design and going straight to coding leads to problems during the integration process. Our simulation environment has to provide a real-world experience and, thus, has to be solidly rooted in such real-world phenomena. Unfortunately, the set of rules of software development is published in a wide variety of media (software engineering journals and conferences, computer-supported collaborative work journals and conferences, books, trade literature, etc.) and no single source exists in which all are compiled. Therefore, the first exercise towards our goal of creating an educational software development simulation environment has been to research, identify, and compile a compendium of the set of fundamental rules of software development. Simultaneously, descriptions of simulation scenarios that would effectively illustrate these rules on both the individual and organizational levels have been created.

As an example of how the simulator facilitates software process learning, consider Brooks' Law. A possible scenario demonstrating this law might be the following:

> *The project is late. The project manager decides to handle this by hiring a few more people, figuring that more manpower will result in a higher productivity. Much to her surprise, each developer's productivity level drops as they spend a significant part of their work hours meeting and communicating with each other, trying to bring the new employees up to speed. She also notices that the productivity levels of the new employees are significantly lower than those of the trained employees. All of this results in an overall decrease in team productivity and an even later project.*

The user going through this scenario would be able to make an assessment of the situation (the project is late), make a decision and perform a subsequent action (hire more people), experience the consequences of his action, both short-term (lower team productivity levels) and long-term (late project), and see the underlying, non-obvious causes of those consequences (developers spending more time communicating and less time developing software). As a result, the individual would gain a concrete, experiential understanding of this important, yet complex concept.

Note that this particular example shows results at both the individual and organizational level. An individual, in this case the manager, is able to understand the consequences of her action. Similarly, however, the organization as a whole learns: if upper management pressured the manager into hiring more personnel, the manager would be able to provide concrete evidence that this may not be a wise decision. Similarly, if the manager encountered a colleague in the same situation as her (a late project and considering hiring more people), she would be able to share the lesson she learned. Thus, even if an individual uses the simulation environment, the organization as a whole may experiences positive effects from this exercise.

6.2. Models

Any simulation environment is driven by a model of the real world. Rather than constructing one such overarching model, we plan to construct a series of models of incremental complexity, each model based on a subset of the rules identified in Section 6.1. Simpler models can be used in focused lessons to highlight small sets of rules that illustrate particular issues. Models of more complexity, constructed out of larger sets of these general software development rules, can be used to illustrate the difficulty of the overall process of software development. When built on rules that embody a specific organization's software process, the simpler models can be used to teach particular aspects of the organization's process to the individual in training. Complex models built on organization-specific rules can be used both to teach the overall process to the individual and to aid the organization in viewing the software process in such a way as to discover areas for improvement. In general, the use of models allows tailoring of the simulation environment to portray organizations' unique software processes and to provide different lessons as part of different simulation runs.

In creating the model, one particular difficulty we anticipate involves the encoding of the software engineering rules. Several questions about the parameterization of the model must be addressed: a) what are the constraints and the variables whose values must obey those constraints; b) what are the constants that influence the values of those variables; c) what are the equations that embody the cause and effect rules determining the behavior of the model; d) how are the (often conflicting) overall goals of software engineering and the individual goals of each entity involved in the simulation encoded into the model?

As an example, consider the following simulation scenario that illustrates the software engineering "law" which says that skipping the design phase leads to highly problematic integration:

> *The developers proceed directly from the requirements phase to implementation, skipping the design phase completely. When they begin to integrate, the error rate of the software skyrockets, the quality of the software drops dramatically, and each developer's mood plummets. They must spend several months (while the cost meter is ticking away) integrating all of the different developers' pieces of code before the system works.*

Expressed qualitatively, this situation is easily described and well understood. However, in order to make this scenario executable in a simulator, a quantitative representation of its behavior, including mathematical equations describing the relationships between all of the different variables and factors involved, is needed. For instance, exactly how many person-months longer does development take when the design phase is skipped? Precisely how many more bugs are present in a piece of software that was developed without a design phase than one that was thoroughly designed before it was implemented? How much does each developer's motivation actually drop as the result of such a situation, and how, in turn, does this affect the resulting productivity of the team? In essence, an exact schema with which to evaluate the precise cost of each action the player can take must be adopted. We intend to leverage information from sources such as COCOMO [5] in creating models that are as close to the real world as possible, neither overplaying nor underplaying the effects portrayed in the simulation.

6.3. Simulation Environment

The most important feature of our proposed simulation environment is that it will be graphical. Learning through visual clues has proven to be far advantageous over simply studying textual output [25] and our simulation environment intends to take full advantage of this fact. For example, consider the rule that adding personnel to a project that is already late typically makes that project later [6]. A simulation environment like SESAM, which is text-based, will show the effect of this rule in the eventual outcome when a student decides to add people to a late project: the project indeed will be later [10]. However, it is unclear as to *why* the project may be later. This is where our simulation environment will take full advantage of its visual front-end: when a student decides to add people to a project that is late, the simulation environment will graphically show that the number of meetings (both face-to-face and as a group) increases. Specifically, it will show personnel assembling in meeting rooms at a greater frequency; it will show complaints from existing personnel to the project manager that they cannot get their work done due to these meetings; and it will show that the project is delivered at a later date. In effect, the graphical nature of our simulation environment allows us to clearly illustrate cause and effect, rather than effect only.

A second important characteristic of our proposed simulation environment is that it is interactive and, as such, supports a student playing different roles. This kind of interactivity is required to engage a student as much as possible in the learning process—students continuously can influence and steer the simulation to experiment with different decisions leading to different situations and outcomes. Being able to play different roles in this process allows a student to examine the effects of a decision made in one particular role on the job and responsibilities of another. For example, in playing a project manager a student may decide that, due to cost considerations, coding has to be performed with fewer personnel. Switching to the role of a coder, then, will allow a student to see such effects as the coder receiving more tasks under heavier time-pressure and consequently having to work overtime to get all work finished. Although the student as a project manager could see the overall effect, the ability to switch roles into that of a coder provides a more detailed, hands-on illustration that likely will increase the learning factor significantly because of the direct nature of its feedback.

Interactivity in our simulation environment goes beyond the interactivity traditionally found in games. Whereas time always continues in games, thereby forcing a player to either let the simulation progress as is or to make decisions under relative strict time pressure, the focus on education that is present in our simulation environment requires advanced facilities to stop, examine, rollback, and continue simulations. Saving the state of a simulation provides rudimentary support for doing so but is cumbersome to use effectively; more advanced support is necessary to encourage a student to fully explore alternative scenarios. In particular, we plan to base our simulation environment on parallel timelines: each timeline represents a different alternative (as chosen by a user) that evolves independently. By carefully differentiating the critical parameters of each timeline, a student can in effect perform multiple simulations at once and study the effects of varying certain parameters while keeping other parameters exactly the same. It should be noted that a student will be able to go back to any point in time on one or all of the timelines to examine—or even roll back, change, and continue—any number of ongoing simulations. This feature is particularly powerful for organizational learning: an organization can quickly examine multiple different paths.

Although essential, the fact that our simulation environment is graphical and interactive represents a rather straightforward design problem. However, several other issues complicate this design problem considerably. Specifically, our simulation environment needs to be based on careful tradeoffs among such considerations as faithfulness to reality, level of detail, usability, teaching objectives, and "fun factor". For example, the more detail that is provided in the simulation and the more faithful the actual simulation is to the real world, the harder it may be for students to extract from the simulation those rules that they are supposed to learn: too many cause and effect relationships may be occurring in parallel for the simulation to be educationally useful. As another example, educational purposes and fun factors may call for visual feedback effects to be "over the top" such that it can be easily recognized by students that something is wrong. However, this would directly contradict the issue of faithfulness to reality.

Clearly, a delicate balance has to be struck among all the design parameters. Ideally, even, the simulation environment supports a certain amount of configurability that allows each design parameter to vary per simulation.

As a final note, we will base our environment on lessons learned by such games as SimCity [11] and The Sims [12] in providing the desired level of functionality while maintaining a graphical and entertaining environment in which users can learn effectively. These games have succeeded in doing so, and not surprisingly we aim for the same level of success. We anticipate being able to leverage existing generic simulation engines such that we will be able to concentrate on building the actual graphical environment through which users will interact and the interaction of the environment with the models underneath.

7. Conclusions

We have identified and sketched a method for teaching software development processes—both at the individual and organizational levels. Our belief is that a game-based software development simulation environment can effectively facilitate software development process learning at both of these levels. Individuals can use such an environment to create an understanding of the various dynamics that underlie both their organization's unique software process and software development in general. Organizations can use such an environment to provide them with an overall view of their process, allowing critical examinations to discover possible areas for process improvement, and specific paths through which such improvements can be made. We are currently in the process of realizing this vision. In particular, we have assembled a large set of rules of software engineering and are in the process of encoding these into several different models. In parallel, we have started to design our simulation environment and are currently investigating the possibility of reusing existing simulation infrastructure to make the development effort simpler.

Acknowledgements

We would like to thank Compaq for their support of this research, Eric Dashofy for valuable feedback on the idea of using simulation for software engineering education, and the anonymous reviewers for the many helpful suggestions that have been incorporated in this paper.

References

1. *Hogwarts MOO.* 1999. dune.net:7500
2. *LostAges.* 1999. mudz.org:6969
3. *UtopiaMOO.* 1998. donald-duck.ele.tue.nl:1111
4. Abdel-Hamid, T., *Lessons Learned from Modeling the Dynamics of Software Development.* Communications of the ACM, 1989. **32**(12): p. 1426-1438.
5. Boehm, B.W., et al., *Cost Models for Future Software Life Cycle Processes: COCOMO 2.0*. 1995, University of Southern California.
6. Brooks, F.P., *The Mythical Man-Month: Essays on Software Engineering*. 2 ed. 1995: Addison-Wesley. 336.
7. Chua, Y.S. and C. Winton, *A Simulation Tool for Teaching CPU Design and Microprogramming Concepts*, in *Conference Proceedings on APL as a Tool of Thought*. 1989, ACM. p. 94-100.
8. Collofello, J.S., *University/Industry Collaboration in Developing a Simulation Based Software Project Management Training Course*, in *Proceedings of the Thirteenth Conference on Software Engineering Education and Training*, S. Mengel and P.J. Knoke, Editors. 2000, IEEE Computer Society. p. 161-168.
9. Dawson, R., *Twenty Dirty Tricks to Train Software Engineers*, in *Proceedings of the 22nd International Conference on Software Engineering*. 2000, ACM. p. 209-218.
10. Drappa, A. and J. Ludewig, *Simulation in Software Engineering Training*, in *Proceedings of the 22nd Internation Conference on Software Engineering*. 2000, ACM. p. 199-208.
11. Electronic Arts, *SimCity 3000*. 1998.
12. Electronic Arts, *The Sims*. 2000.
13. Electronic Arts, *Ultima Online Renaissance*. 2000.
14. Kusumoto, S., et al., *A New Software Project Simulator Based on Generalized Stochastic Petri-net*, in *Proceedings of the 1997 International Conference on Software Engineering*. 1997, ACM. p. 293-302.
15. Law, A.M. and W.D. Kelton, *Simulation Modeling and Analysis*. 3 ed. 2000: McGraw-Hill Companies, Inc.
16. Lindheim, R. and W. Swartout, *Forging a New Simulation Technology at the ICT.* IEEE Computer, 2001. **34**(1): p. 72-79.
17. Lucas Arts Entertainment Company, *Escape From Monkey Island*. 2000.
18. Lyra Studios, *Underlight*. 2000.
19. Madachy, R., *System Dynamics Modeling of an Inspection-Based Process*, in *Proceedings of the Eighteenth International Conference on Software Engineering*. 1996, IEEE Computer Society.
20. Mattel Interactive, *realMYST*. 2000.
21. Raffo, D., *Modeling Software Processes Quantitatively and Assessing the Impact of Potential Process Changes*

on Process Performance, in *Graduate School of Industrial Administration*. 1996, Carnegie Mellon University: Pittsburgh, PA.

22. Rolfe, J.M., *Flight Simulation (Cambridge Aerospace Series)*. 1988: Cambridge University Press.
23. Rus, I., J.S. Collofello, and P. Lakey, *Software Process Simulation for Reliability Management.* The Journal of Systems and Software, 1999. **46**(3): p. 178-182.
24. Samu Games, *Artifact*. 2000.
25. Shneiderman, B., *Designing the User Interface: Strategies for Effective Human-Computer Interaction*. 2nd ed. 1992: Addison-Wesley Publishing Company.
26. Sony Online Entertainment, *Everquest*. 1999.