# Design and Evaluation of an Educational Software Process Simulation Environment and Associated Model

Emily Oh Navarro and André van der Hoek
*Department of Informatics*
*Donald Bren School of Information and Computer Sciences*
*University of California, Irvine*
*Irvine, CA 92697-3425 USA*
*emilyo@ics.uci.edu, andre@ics.uci.edu*

***Abstract***

*Simulation is an educational tool that is commonly used to teach processes that are infeasible to practice in the real world. Software process education is a domain that has not yet taken full advantage of the benefits of simulation. To address this, we have developed SimSE, an educational, interactive, graphical environment for building and simulating software engineering processes in a game-like setting. We detail the design of SimSE, present an initial simulation model of a waterfall process that we developed, and describe an experiment that we conducted to evaluate the educational potential of SimSE and its initial model.*

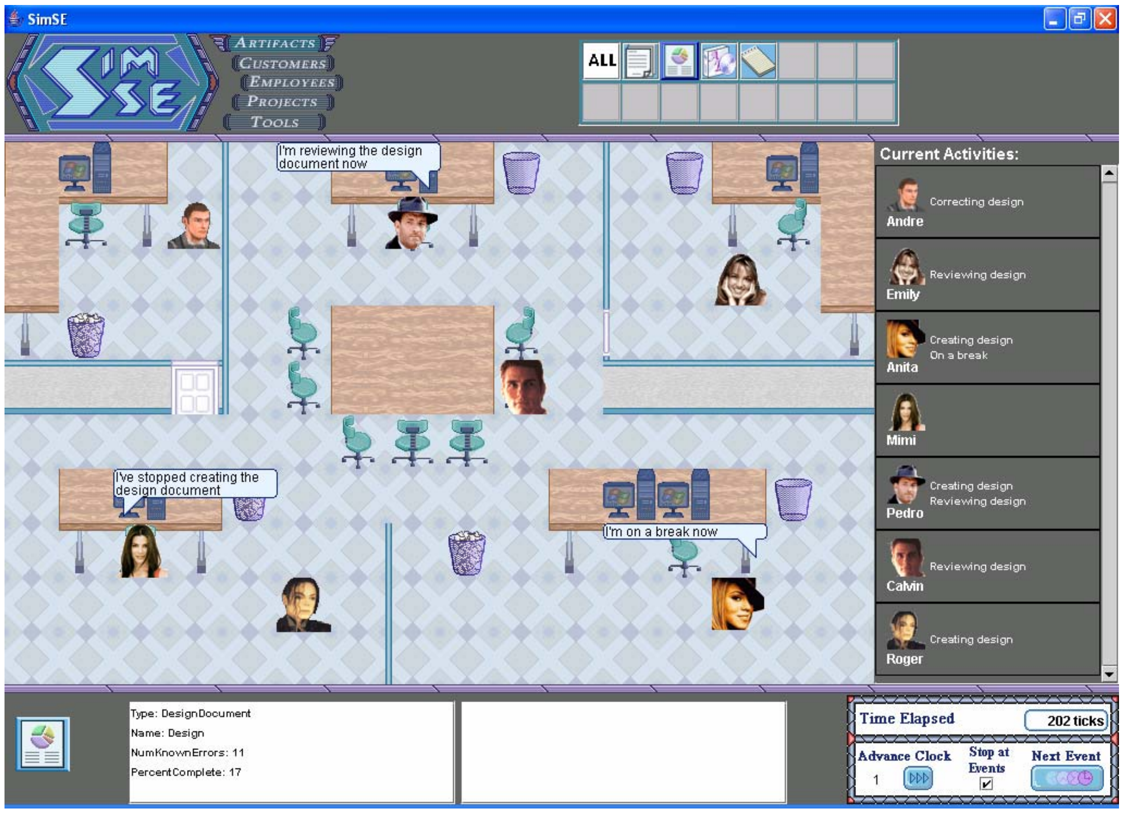
## 1. Introduction

Simulation is a powerful educational tool that is widely used in a number of different domains [10, 15, 19] to gain valuable hands-on experience of a process being simulated without any of the potential monetary costs or harmful effects that may result from actual real-world experience. As a result, students are free to repeat experiences, experiment with different approaches, and push boundaries, gaining deeper insights with each simulation run.

Although software process education would be an ideal domain in which to leverage the benefits of simulation, it that has not yet taken full advantage of this approach. There have been a few exceptions [5, 7, 13, 14, 17] that have identified promising avenues, but have not yet fully pushed the boundaries of simulation in software engineering education. Namely, these approaches have been limited in one or more of the following areas: interactivity, customizability, and/or graphics.

We build on the knowledge gained by these approaches with SimSE, a new educational, graphical, game-based software engineering simulation environment that explicitly addresses the shortcomings of existing educational software engineering simulations in two critical ways: First, SimSE is fully graphical and interactive, providing a game-like environment in which to "play" software engineering that makes learning fun for the students and hence, more effective [8]. Second, SimSE provides a model builder tool that simplifies the process of building simulation models by hiding the underlying process modeling language from the user, allowing them to build models at a higher level of abstraction.

SimSE is a single-player game in which the player takes on the role of project manager and must manage a team of developers in order to successfully complete (a particular aspect of) an assigned software engineering project. The player drives the process by, among other things, hiring and firing employees, assigning tasks to them, monitoring their progress, and purchasing tools. At the end of the game the player receives a score indicating how well they performed, and additional information that was hidden throughout the game is revealed to give the player some insight into why they were given the score they received.



I'm reviewing the design document now

I've stopped creating the design document

I'm on a break now

Current Activities:

| | |
|---|---|
| Andre | Correcting design |
| Emily | Reviewing design |
| Anita | Creating design<br>On a break |
| Mimi | |
| Pedro | Creating design<br>Reviewing design |
| Calvin | Reviewing design |
| Roger | Creating design |

Type: DesignDocument
Name: Design
NumKnownErrors: 11
PercentComplete: 17

Time Elapsed 202 ticks

Advance Clock 1

Stop at Events ✓

Figure 1: SimSE Graphical User Interface.

Because having an engaging and interactive graphical user interface is considered essential to any successful educational simulation [9, 18], the user interface of SimSE is fully graphical, as shown in Figure 1. The center part of the interface displays a virtual office in which the software engineering process takes place, including typical office surroundings (e.g., desks, chairs, computers, meeting rooms) and employees. Employees “communicate” with the manager (player) through pop-up speech bubbles over their heads, in which they inform the player of important information, such as when they have started or completed a task, or to express a response to one of the player’s actions.

SimSE has a variety of control mechanisms for playing the game: The user drives the simulation through the controls of the clock in the lower right corner. They can interact with the employees through right-click menus (not shown) on each employee. Detailed information about each object (artifacts, customers, employees, projects, and tools) can be obtained by clicking on the corresponding tab for the object’s type in the upper left hand corner of the interface, and then choosing the image representing the desired object. Finally, the player can see all of the activities in which each employee is currently participating on the right side of the interface. More information on the interaction and game play of SimSE can be found in its user manual, which is available on the SimSE web site (see “More Information” at the end of the paper).

The remainder of this paper details SimSE. Section 2 provides a brief description of SimSE’s architecture and presents our process modeling approach. Section 3 describes a simulation model of a waterfall process that we developed for use in SimSE, including an example game play scenario of a simulation using that model. In Section 4 we describe an experiment we performed to evaluate the teaching potential of SimSE and the waterfall simulation model, and we conclude in Section 5.

## 2. Architecture

One of the fundamental goals of SimSE is to support customization of the software process models it simulates. Real-world software processes vary with different application domains, organizations, and cultures, and therefore SimSE must be able to portray a wide range of processes as well. Furthermore, instructors using SimSE may belong to different schools of thought regarding best software engineering practices, and may have different teaching objectives that require different types of models. SimSE was designed to allow this customization, as can be seen from its architecture shown in Figure 2. An instructor uses the model builder tool to create a simulation model that embodies the process and lessons they wish to teach their students. (For more information on the modeling approach and model builder tool, see [12].) The generator component interprets this model and automatically generates Java code for a state management component, a rule and action execution component, and the graphical user interface, which are inserted into the generic simulation environment. A student uses this custom-generated environment to practice the situations captured by the model.

To get some idea about what goes in a SimSE model, we list its major components here:

- *Object Types*: These are the templates that each major entity participating in the simulation must instantiate. Every object type consists of a name and a set of typed attributes, and is of one of five *meta-types*: Employee, Artifact, Tool, Project, or Customer.
- *Start State*: The start state is the set of objects that are present when a simulation begins. Each start state object instantiates one of the object types defined in the object builder, and assigns starting values to all of its attributes.
- *Actions*: These are the activities in which objects in the simulation can participate. For each action, the following information is specified: a name; one or more participants, action triggers, and action destroyers. An action trigger refers to the set of conditions that cause the action to begin in the simulation, while a destroyer is what causes it to end.
- *Rules*: Rules are attached to actions and define the effect of an action. We distinguish three types of rules in a SimSE model: *create objects rules*, *destroy objects rules*, and *effect rules*. As its name indicates, a create objects rule causes new objects to be created. Conversely, the firing of a destroy objects rule results in the destruction of existing objects. An effect rule specifies the complex effects of an action on its participants' states, including the values of their attributes and their participation in other actions.
- *Graphics*: An image must be assigned to each object in the start state. This image will be used to represent the object in the graphical user interface of the simulation. In addition, the layout of the "office" must be specified.

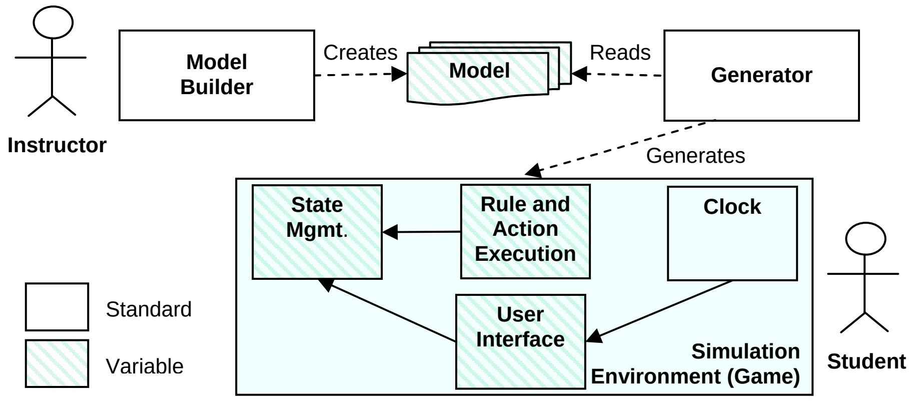


Figure 2: SimSE Architecture.

## 3. Example Waterfall Model

As an initial attempt at building an educational software process model, we developed a model of the waterfall life cycle. Although the waterfall is not the most interesting or challenging life cycle model that exists, it is still commonly taught, and its simplicity allows us to demonstrate the principles of the environment and teach some lessons about the software engineering process in general. Clearly, more models of various size and complexity need to be built and evaluated, which we are currently in the process of doing (see Section 4.2). Here we detail the lessons we aimed to teach in designing the model, and then present an example of how a typical simulation of this model might be played.

### 3.1 Goals

Because this particular model was purposed to emulate a waterfall process, we developed the model to reward the player for following the proper steps and practices of the waterfall model and penalize them for doing otherwise. In parallel, we aimed to teach a number of overall lessons about the software engineering process in general. These lessons were taken from a compendium of 86 "fundamental rules of software engineering" [11] that we gathered by surveying software engineering literature, talking to software engineering professionals, and perusing lecture notes from the introductory software engineering class at UC Irvine. The following are a few examples of the lessons that are implemented in our model.

- *Do requirements, followed by design, followed by implementation, followed by integration, followed by testing.*
- *At the end of each phase, perform quality assurance activities (e.g., reviews, inspections), followed by correction of any discovered errors.*
- *If you do not create a high quality design, integration will be slower and many more integration errors will be introduced.*
- *Developers' productivity varies greatly depending on their individual skills, and matching the tasks to the skills and motivation of the people available increases productivity* [2, 4, 16].
- *The greater the number of developers working on a task simultaneously, the faster that task is finished, but more overall effort is required due to the growing need for communication among developers* [3].
- *Software inspections are more effective the earlier they are performed* [20].
- *The better a test is prepared for, the higher the amount of detected errors.*
- *Monetary incentives increase motivation, which leads to increased productivity (but faster expenditures)* [20].
- *The use of software engineering tools leads to increased productivity* [20].
- *New requirements frequently emerge during development since they could not be identified until portions of the system had been designed or implemented* [6].

In addition to these, there are a number of other general workplace issues not specific to software engineering that are included in the model. For instance, employees sometimes get sick, take breaks when they are tired, become less productive when they are tired, and quit when they are upset about something significant (e.g., a pay cut).

### 3.2 Game Play Example

To illustrate how a few of these lessons are exhibited during game play, as well as to provide an example of what a SimSE simulation game is like, we will use a brief scenario of how a student may use SimSE in completing the task of developing a software product from requirements specification to product delivery.

When the game begins, the player sees a starting narrative that describes to them the goals of the simulation. In this example, the starting narrative is the following: ”*Welcome to SimSE! Your task is to create Groceries@Home, a Web-based system that will allow people to place orders over the Internet for groceries to be delivered to their homes. The customer is the Grocery Home Delivery Service, a company who, up until now, has taken orders for groceries solely by telephone, but now wants to step into the information age. Your budget is $280,000, and you have 1,350 clock ticks to complete the project. However, you should keep checking your project info to monitor this information – the customer has the tendency to introduce new requirements, and will sometimes give you more time and/or money along with those new requirements. Your final score will be out of 100 points, and it will be calculated based on how complete and error-free your code is, whether your code is integrated or not, and how well you stick to your budget and schedule. Good luck!*”

The first step this player takes is to go through some of their resources and assess what they have to work with. The player brings into focus each employee and views his or her skill levels in each area, noting who is good and bad at the different tasks, and then looks at each tool and notes its cost. The player sees that two of the tools, the JUnit automated testing tool and the Eclipse IDE, have a cost of zero, so they immediately "purchase" those two.

Since this is the waterfall model, the player decides to start out having their employees specify the requirements for the product. Because the employees Anita, Calvin, Pedro, and Andre (see Figure 1) have the most experience in requirements, these are the ones the player assigns to start creating the requirements. The player then steps the simulation forward 20 clock ticks, after which they see that the requirements document is 7% complete. Now that some requirements have been specified, some of the other employees can start reviewing them. The player assigns the rest of the employees, Mimi, Roger, and Emily, to review the requirements document. The player steps forward 20 more clock ticks, and sees that the requirements document is now 14% complete, and the reviewers have discovered three errors. At this point the player is thinking that requirements specification is going a bit slower than they would like, so they decide that it might be worth the $10,000 to purchase the requirements capture tool, which they do. They then step forward another 20 clock ticks, and see that the requirements document is now 25% complete, and are pleased that their purchase seems to have sped things up by a factor of about 1.5.

We now fast forward a bit, and assume that the employees finished the requirements document, reviewed it, and the player has their requirements experts, Anita, Calvin, Pedro, and Andre correcting it. The player now decides to move on to the design phase. Since the requirements tool seemed to be so helpful, the player also purchases a design environment tool for $5,000. Unfortunately, two of the three experts in design (Andre and Anita) are also requirements experts, so they are already engaged in correcting the requirements document. As a result, the player assigns only one employee who is experienced in design (Emily) along with two less-experienced designers, Roger and Mimi, to start creating the design document. They continue this until the other employees are finished correcting the requirements document. At this point, the designers have been designing for 33 clock ticks, and they are only 4% finished. Now that the other two expert designers, Andre and Anita, are freed up, the player adds them to the designing task, and has all of the other employees start reviewing the design. The player then steps forward 20 clock ticks and is pleased to see that this reallocation of tasks has sped up design tremendously—the design document is now 10% complete.

The player continues like this until the design document is 100% complete. They then (unwisely) figure that since they had such qualified people working on the design, and they would like to try to finish the project as quickly as possible, they will forgo review and correction of the design document, moving on to the coding phase. They assign all of the coding experts to coding, and they complete the code. Once the player begins inspection, however, they realize that they have made a bad decision somewhere, because inspection seems to be endless, taking 230 clock ticks and finding 152 errors. (Many of these are errors were carried over from the design document, which the player will find out later.)

The player has their employees correct all of these errors, and then integrate the code (which also seems to be awfully slow). They then prepare the system test plan, test the system, which reveals 108 errors, and correct these errors. Due to all of the errors in the code, which required extra time spent on inspection, testing and correction, the project is now slightly late (145 more clock ticks than allotted) and $20,580 over budget. The player delivers the product to the customer, and receives a score of 81 out of 100—not a bad score, but it could have been better.

To allow players to understand what they did wrong, SimSE reveals any hidden attributes after the completion of a game. In this case, the player discovers that there were 49 unknown errors in the design document, indicating that they probably should have had the employees review and correct the design before moving on to coding. Clearly, multiple simulation runs of the same model are needed for the player to try to correct their mistakes and explore different approaches, in order to truly understand the lessons and the process being taught.

## 4. Evaluation

As an initial evaluation of the SimSE simulation environment and the waterfall simulation model we developed, we conducted an experiment in which we had undergraduate computer science students at UC Irvine play the game and provide us with their feedback. This section describes the design and results of this experiment.

### 4.1 Experiment Design

We recruited 29 students who had passed the introductory software engineering course at UC Irvine to participate in the experiment. They first received instruction on how to play SimSE, and then played the game for approximately two hours, completing one to two games. Following this, they completed a questionnaire stating their thoughts and feelings about the game in general, their opinions about the pedagogical effectiveness of the game in teaching software engineering process issues, and their educational and professional background in software engineering. Some of these questions asked for a numerical answer on a one to five scale, while others allowed students to write out their responses in free form.

### 4.2 Experiment Results

In general, students' feelings about the game were favorable, as summarized in Table 1. On average, students found the game enjoyable to play (3.5 rating out of 5) and relatively easy to play (3.2). They also felt that it was quite successful in reinforcing software engineering process issues taught in the introductory software engineering course they had taken (3.7) and equally successful in teaching software engineering process issues in general (3.6). For the most part, they agreed that SimSE would be helpful to teaching software engineering concepts if incorporated into the introductory software engineering course (3.5).

Students' answers to the open-ended questions also reflected their positive feelings about SimSE. Regarding the enjoyability of the game, some students remarked: *"It does a good job of reinforcing the process in a very fun way!"*, *"[It] makes [software engineering] seem more real and makes it more enjoyable."*, and *"It is a unique way to learn and study software engineering."* Regarding how well the game teaches software engineering process issues, students wrote ("52" refers to the introductory software engineering course): *"[My favorite aspect of* *the game was] managing a team of employees. It is cool to see how they react to certain environments, and see how the project develops according to the selection of employees for different jobs.”*, *“[It taught me] delegating tasks and budgeting. In 52 we learned how to create but not manage.”*, *“52 teaches the intellectual level, overall view; the game illustrates this by feel and trial/error.”*, and *“[Having to deal with] pay, energy, and mood introduces more complex, real-life issues present in a workspace.”*

Table 1: Questionnaire Results.

| ***Question*** | ***1*** | ***1.5*** | ***2*** | ***2.5*** | ***3*** | ***3.5*** | ***4*** | ***4.5*** | ***5*** | **Avg** |
|---|---|---|---|---|---|---|---|---|---|---|
| How enjoyable? (1=least enjoyable, 5=most enjoyable) | 1 | 0 | 1 | 0 | 12 | 2 | 10 | 0 | 3 | **3.5** |
| How difficult/easy? (1=most difficult, 5=easiest) | 0 | 0 | 7 | 0 | 9 | 1 | 11 | 0 | 1 | **3.2** |
| Reinforces material taught in class? (1=not at all, 5=definitely) | 0 | 0 | 2 | 1 | 9 | 1 | 8 | 2 | 6 | **3.7** |
| Teaches new process knowledge? (1=not at all, 5=definitely) | 3 | 0 | 14 | 0 | 6 | 0 | 4 | 0 | 1 | **2.5** |
| Teaches SE process in general? (1=not at all, 5=very much so) | 0 | 0 | 2 | 0 | 12 | 1 | 10 | 0 | 4 | **3.6** |
| Incorporate into SE course? (1=not at all, 5=very much so) | 0 | 0 | 3 | 0 | 12 | 2 | 6 | 1 | 5 | **3.5** |
| As an optional part? (1=not at all, 5=very much so) | 0 | 0 | 6 | 1 | 8 | 1 | 7 | 0 | 6 | **3.4** |
| As a mandatory part? (1=not at all, 5=very much so) | 1 | 0 | 6 | 0 | 9 | 0 | 8 | 1 | 4 | **3.3** |

Although responses were positive for the most part, it is clear that some aspects of the game need to be improved. The most negative response was that students did not feel that the game taught them much *new* software process knowledge (2.5). While reinforcing the concepts taught in lecture is useful in and of itself, the game would be even more useful if it could teach new concepts. It is understandable, however, that this particular model did not teach much new knowledge, since it was based on the waterfall model, which is a simple model that is frequently talked about in lectures. We believe that building additional models of different sizes, scopes, and foci will ameliorate this weakness. We are currently in the process of testing this hypothesis by building three new models: one that teaches the Rational Unified Process [9], one that teaches Extreme Programming [1], and a third, more detailed model that teaches the inspection process. Other models are in the planning stages.

Some of the students also wished that they were given more explanation as to why they received their score and felt that it was sometimes hard to tell where they went wrong. To address this, we plan to develop an explanatory tool as part of the simulation environment that will provide students with a log of events, a record of which rules were executed at which times, and a trace of the different objects’ attributes over time, so they can clearly see which of their decisions had which effects. Aside from this, the other aspects of the game that the students were unhappy with were mainly technical issues, such as the lack of a “stop” button when the clock was ticking, and the quality of the graphics. We are currently in the process of addressing these issues.

## 5. Conclusions and Future Work

We have presented SimSE, an educational, graphical, interactive, simulation environment that allows instructors to model software engineering processes and allows their students to play, practice, and learn these processes in an engaging and effective manner, without the time and scope constraints of the academic environment. The experiment we performed suggests that SimSE would be a valuable addition to an introductory software engineering course to illustrate and enforce the lessons taught in lectures.

We are currently building more simulation models, designing an explanatory tool, and making various enhancements to the simulation environment. We will perform more experiments in order to asses the educational effectiveness of the models, the refined simulation environment, and the planned explanatory tool. SimSE will also be incorporated into actual class use at UC Irvine during spring quarter 2005, and will be tested at other institutions in the near future as well.

## Acknowledgements

Effort partially funded by the National Science Foundation under grant numbers DUE-0341280, CCR-0093489 and IIS-0205724. More information about SimSE can be found at http://www.ics.uci.edu/~emilyo/SimSE.

## References

[1] K. Beck, *Extreme Programming Explained: Embrace Change*. Reading, MA: Addison-Wesley, 2000.

[2] B. W. Boehm, *Software Engineering Economics*. Upper Saddle River, NJ: Prentice Hall, Inc., 1981.

[3] F. P. Brooks, *The Mythical Man-Month: Essays on Software Engineering*, 2 ed. Boston, MA: Addison-Wesley, 1995.

[4] G. E. Bryan, "Not All Programmers are Created Equal," in *Software Engineering Project Management*, R. H. Thayer, Ed. Los Alamitos, CA: IEEE Computer Society, 1997, pp. 346-355.

[5] J. S. Collofello, "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, Eds.: IEEE Computer Society, 2000, pp. 161-168.

[6] B. Curtis, H. Krasner, and N. Iscoe, "A Field Study of the Software Design Process for Large Systems," *Communications of the ACM*, vol. 31, pp. 1268-1287, 1998.

[7] A. Drappa and J. Ludewig, "Simulation in Software Engineering Training," in *Proceedings of the 22nd International Conference on Software Engineering*: ACM, 2000, pp. 199-208.

[8] M. Ferrari, R. Taylor, and K. VanLehn, "Adapting Work Simulations for Schools," *The Journal of Educational Computing Research*, vol. 21, pp. 25-53, 1999.

[9] K. L. Higbee, "Recent Research on Visual Mnemonics: Historical Roots and Educational Fruits," *Review of Educational Research*, vol. 49, pp. 611-629, 1979.

[10] R. Lindheim and W. Swartout, "Forging a New Simulation Technology at the ICT," *IEEE Computer*, vol. 34, pp. 72-79, 2001.

[11] E. O. Navarro, ""The Fundamental Rules" of Software Engineering," http://www.ics.uci.edu/~emilyo/SimSE/se_rules.html, 2002.

[12] E. O. Navarro and A. van der Hoek, "Software Process Modeling for an Educational Software Engineering Simulation Game," *Software Process -- Improvement and Practice*, 2005 (in press).

[13] U. Nulden and H. Scheepers, "Understanding and Learning about Escalation: Simulation in Action," in *Proceedings of the 3rd Process Simulation Modeling Workshop (ProSim 2000)*. London, United Kingdom, 2000.

[14] D. Pfahl, M. Klemm, and G. Ruhe, "Using System Dynamics Simulation Models for Software Project Management Education and Training," in *Proceedings of the 3rd Process Simulation Modeling Workshop (ProSim 2000)*. London, United Kingdom, 2000.

[15] J. M. Rolfe, *Flight Simulation (Cambridge Aerospace Series)*. Cambridge, UK: Cambridge University Press, 1988.

[16] H. Sackman, W. J. Erikson, and E. E. Grant, "Exploratory Experimental Studies Comparing Online and Offline Programming Performance," *Communications of the ACM*, vol. 11, pp. 3-11, 1968.

[17] H. Sharp and P. Hall, "An Interactive Multimedia Software House Simulation for Postgraduate Software Engineers," in *Proceedings of the 22nd International Conference on Software Engineering*: ACM, 2000, pp. 688-691.

[18] B. Shneiderman, *Designing the User Interface: Strategies for Effective Human-Computer Interaction*, 2nd ed. Boston, MA: Addison-Wesley, 1992.

[19] D. Skrien, "CPU Sim 3.1: A Tool for Simulating Computer Architectures for Computer Organization Classes," *ACM Journal of Educational Resources in Computing*, vol. 1, pp. 46-59, 2001.

[20] J. D. Tvedt, "An Extensible Model for Evaluating the Impact of Process Improvements on Software Development Cycle Time," Ph.D. Dissertation, Arizona State University, 1996.