SIMSE: AN INTERACTIVE SIMULATION GAME FOR SOFTWARE ENGINEERING EDUCATION

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

Abstract

The typical software engineering education lacks a practical experience of the *process* of software engineering—students are presented with relevant process theory in lectures, but have limited opportunity to put these concepts into practice in an associated class project. SimSE is an educational, interactive, fully graphical computer game that simulates software engineering processes, and is designed specifically to train students in situations that require an understanding and handling of software process issues. In this paper we describe SimSE, including its educational goals, its design, and its implementation.

Key Words

Software engineering education, educational games, software engineering simulation, simulation games

1. Introduction

While the software industry has had remarkable success in developing software that is of an increasing scale and complexity, it has also experienced a steady and significant stream of failures. Most of us are familiar with public disasters such as failed Mars landings, rockets carrying satellites needing to be destroyed shortly after takeoff, or unavailable telephone networks, but many more "private" problems occur that can be equally disastrous or, at least, problematic and annoying to those involved. Examining one of the prime forums documenting these failures, the Risks Forum [1], provides an illuminating insight: a majority of documented failures can be attributed to software engineering *process breakdowns*. Such breakdowns range from individuals not following a prescribed process (e.g., not performing all required tests, not informing a colleague of a changed module interface), to group coordination problems (e.g., not using a configuration management system to coordinate mutual tasks, not being able to deliver a subsystem in time), to organizations making strategic mistakes (e.g., choosing to follow the waterfall process model where a spiral approach would be more appropriate, not accounting for the complexity of the software in a budget estimate). As a result, it is estimated that billions of dollars are wasted each year due to ineffective processes and subsequent faulty software being delivered [2].

We believe the root cause of this problem lies in education: current software engineering courses typically pay little to no attention to students being able to practice issues surrounding the software engineering *process*. The software engineering industry has long recognized the problem of students having insufficient process experience and has repeatedly requested that academia address this problem [3-6]. The underlying issue is time: any class project must be completed within the length of its course. While relevant process theory can be and typically is presented in lectures, the opportunities for students to practically and comprehensively experience the presented concepts are limited. Most course projects simply guide students through a linear execution of the waterfall model (requirements, design, implementation, testing) in which students are left with little discretion. Students cannot decide which overall life cycle model to follow, whether or not to first build a rapid prototype, or even when to set the milestones for their deliverables—these and other decisions are usually made by the instructor. The focus strongly remains on creating project deliverables such as requirements documents, design documents, source code, and test cases, and little room is left to illustrate or experience the principles, pitfalls, and dimensions of the software process. The overall result is that students are unable to build a practical intuition and body of knowledge about the software process, and are ill-equipped for choosing particular software processes, for recognizing potentially troublesome situations, and for identifying approaches with which to address such troublesome situations.

To better prepare students for their future software development jobs and consequent exposures to the software process, we are developing a new approach to teaching and practicing the software process that is complementary to existing software engineering courses. SimSE is a computer-based environment that allows the creation and simulation of software engineering processes. It allows students to virtually participate in a realistic software engineering process that involves real-world components not present in typical class projects, such as teams of people, large scale projects, critical decision-making, personnel issues, multiple stakeholders, budgets, planning, and random, unexpected events. In so doing, it provides students with a platform in which they can experience many different aspects of the software process in a practical manner without the overarching emphasis on creating deliverables. SimSE directly addresses the shortcomings of existing classroom approaches by filling the gap between the process knowledge and theory taught in lectures and the small percentage of this knowledge that students actually put into practice in class projects.

The remainder of this paper is organized as follows: Section 2 outlines the objectives of SimSE as an educational simulation. Section 3 describes the architecture and various components of SimSE in detail. In Section 4 we provide a brief overview of related work, and we end in Section 5 with our conclusions and future directions.

2. Objectives

As an educational simulation, SimSE must make tradeoffs among faithfulness to reality, level of detail, usability, teaching objectives, and “fun factors”. As an example, educational purpose and fun factors may call for visual feedback to be humorous and “over the top”, such that the causal effects of decisions can be easily recognized by students. However, this would contradict the issue of faithfulness to reality. Clearly a delicate balance has to be struck among all design parameters. To guide us in creating this balance, we formulated a set of overarching guidelines for the design of SimSE, based on both the unique nature of software engineering and lessons learned from previous studies about critical success factors for educational simulations:

- *SimSE should illustrate both specific lessons and overarching practices of the software process*. Specific lessons include, among others, Brooks’ Law (adding more employees to a project that is already late will make the project even later, due to the exponentially increased communication overhead [7]) and the fact that the use of a configuration management system normally improves the development process. Overarching practices concern the choice of different life cycle models, the tradeoffs involved in many decisions, and the potential for drastic consequences in case of failure.
- *SimSE should support the instructor in specifying the lessons he or she wishes to teach*. Instructors using SimSE may belong to different schools of thought regarding best software engineering practices, and may have different teaching objectives. Furthermore, real-world software processes vary with different application domains, organizations, and cultures.
- *SimSE should provide a student with clear feedback concerning their decisions*. Consider, for example, a student who hires a plethora of employees in order to rush code development. Later, the student faces a lengthy integration phase with many bugs being discovered. The simulation environment must explain why this situation occurs (e.g., more employees leads to more parallel work, which leads to more integration problems [8]).
- *SimSE should be easy to learn, enjoyable, and comparatively quick.* Without compromising the level of complexity that is needed to make each simulation unique, a single simulation must neither be too difficult nor too long in order to maintain the interest of students and, thus, its value as an educational tool. The enjoyability of the game is a large part of what will make the lessons learned more memorable [9].

In summary, SimSE must be enjoyable and realistic, yet rooted in both conceptual and empirical principles of software engineering. The environment must provide students with a complimentary learning opportunity that reinforces lessons taught in lectures and foreshadows and prepares students for their actual experiences in class projects and future careers.

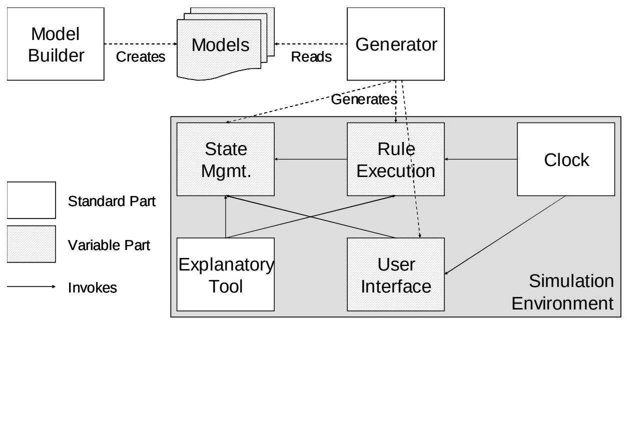
3. Approach

SimSE is a single-player game in which the player takes on the role of project manager of a team of developers. The player must manage these employees to complete a particular (aspect of a) software engineering project. Management activities include, among others, hiring and firing, assigning tasks, monitoring progress, and purchasing tools. In general, following good software engineering practices will lead to positive results while blithely ignoring these practices will lead to miserable failure in completing the project.

3.1 Overall Architecture

Figure 1 illustrates the architecture of SimSE. Models are created using a model builder that allows the specification of employees, artifacts, projects, tools, and customers, as well as activities that these entities can participate in, and rules according to which they behave. Based upon a particular choice of model, a generator interprets the model

Figure 1: SimSE Architecture



and automatically generates code for a state management component, a rule execution component, and the graphical user interface, which are inserted into the generic simulation environment. A student can then use this custom-generated simulation environment to practice the situations captured by the chosen model. Each of the major components in the architecture will be further explained in the following subsections.

3.2 Model Builder and Generator

To facilitate the rapid and relatively easy specification of customized simulation models, a fundamental part of SimSE is its model builder. It is expected that an instructor will use this tool to create models (or alter existing models as necessary) that embody the lessons and processes that they wish to teach to their students. Then, a customized simulation that the students can play will be generated by the Generator component.

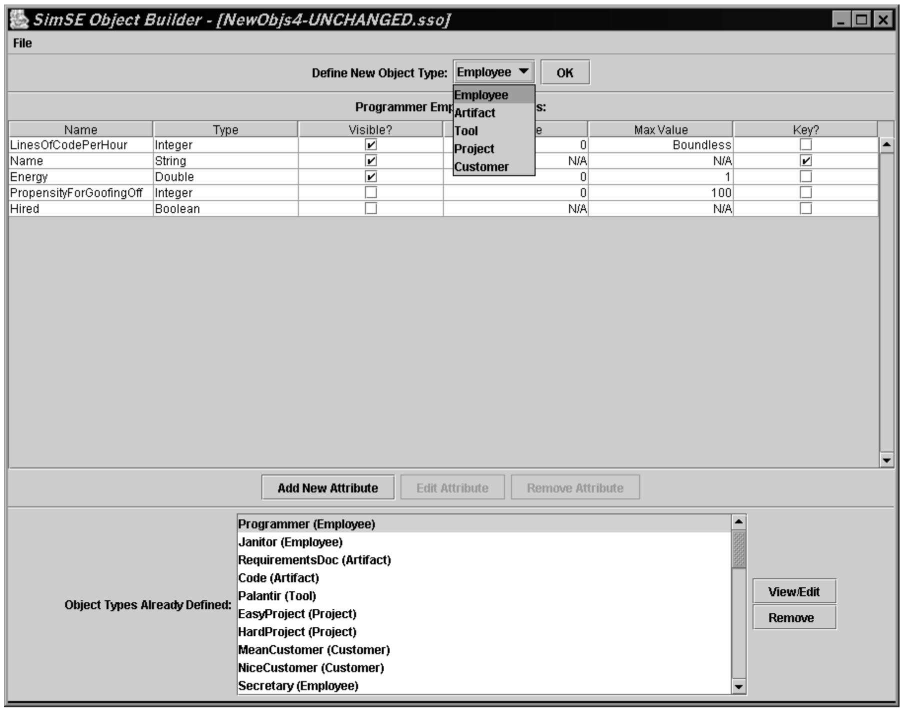
The model builder consists of five parts that correspond to the five parts of a SimSE model: The object builder, the start state builder, the action builder, the rule builder, and the graphics builder.

Object builder. The object builder is used to perform the first step in building a model: defining the object types to be used in the model. Each major entity participating in the simulation will be an instantiation of an object type. Every object type defined must descend from one of five *meta-types*: Employee, Artifact, Tool, Project, or Customer. An object type consists of a name and a set of typed attributes. For example, a Programmer Employee type would be an instance of the Employee meta-type, have the name "Programmer", and may have the following set of attributes: Name (String), Productivity (Double), Experience (Integer), and Hired (Boolean). The user interface for the object builder is shown in Figure 2. For the sake of space, the interfaces for the other builders are not shown, but they are similar in appearance to the object builder in that they all support the modeler in building a model using buttons, drop-down lists, menus, and dialog boxes—no programming is required.

Start state builder. Once the object types for a simulation have been defined, the start state builder can be used to specify the start state for that simulation. The start state refers to the set of objects that are present when the simulation begins. Each one of these objects must be an instantiation of one of the object types defined in the object builder. The start state builder allows the user to create and add an object to the start state by first choosing its object type, and then assigning starting values for all of its attributes.

Action builder. The next part of a SimSE model is the set of actions, or activities in which the objects in the simulation can participate. For example, a "Code" artifact, with one or more "Programmer" Employees and one or more "Compiler" Tools could participate in a "Coding" action, in which the programmers build a piece of code using a compiler. As another example, an Employee could participate in a "break" action, referring to the activity of taking a break, during which he or she rests and does not work. The action builder is the tool that supports defining these actions. It allows the modeler to define actions by specifying the following information for each action: a

Figure 2: Object Builder User Interface.



Name	Type	Visible?	e	Max Value	Key?
LinesOfCodePerHour	Integer	✓	0	Boundless	☐
Name	String	✓	N/A	N/A	✓
Energy	Double	✓	0	1	☐
PropensityForGoofingOff	Integer	☐	0	100	☐
Hired	Boolean	☐	N/A	N/A	☐

Programmer (Employee)
Janitor (Employee)
RequirementsDoc (Artifact)
Code (Artifact)
Palantir (Tool)
EasyProject (Project)
HardProject (Project)
MeanCustomer (Customer)
NiceCustomer (Customer)
Secretary (Employee)

name; one or more participants—roles in the action that can be filled by one or more objects, an action trigger, and an action destroyer. An action trigger refers to the set of conditions that cause the action to begin to occur in the simulation. An action destroyer is similar to an action trigger, but has the opposite effect: whereas a trigger *starts* an action, a destroyer *stops* an action.

Rule builder. After all of the action types have been defined, the next task in building a SimSE model is to attach *rules* to each action type. This is the function of the rule builder. A rule defines an effect of an action—how the simulation is affected when that action is active.

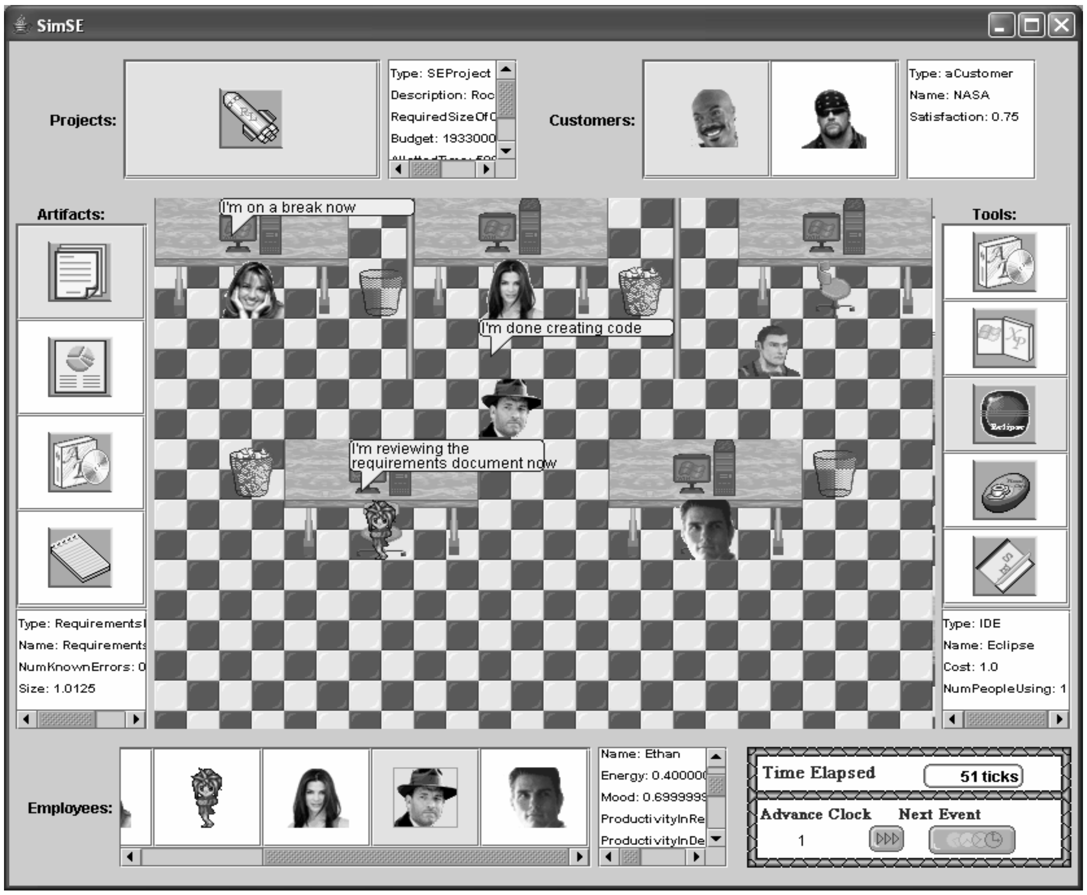
We distinguish three types of rules in a SimSE model: *create objects rules*, *destroy objects rules*, and *effect rules*. As its name indicates, a create objects rule causes new objects to be created in the game. Conversely, the firing of a destroy objects rule results in the destruction of existing 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. For instance, an effect rule attached to the "Coding" action might cause the size of the code to increase by the additive productivity levels of all of the programmers currently working on it. As another example, a "Break" action might have an effect rule attached to it that: a) increases the energy of an employee; and b) deactivates the employee from all other actions he or she is currently participating in for the duration of the "Break" action.

Graphics builder. The final activity in building a SimSE model is specifying the graphics for the simulation. Through the graphics builder, the user can assign an image to each object in the start state, as well as each object that is created by a create objects rule. This image will be used to represent the object in the graphical user interface of the simulation. In addition, the graphics builder includes a map editor that can be used to define the layout of the "office"—starting locations for all of the objects in the game.

3.3 Simulation Environment

The simulation environment is responsible for executing a simulation and creating the user experiences that demonstrate the inherent complexity of the software process. The simulation loop itself operates in the following manner (refer to Figure 1): The clock drives the simulation by emitting ticks at equal time intervals. At every clock tick, the rule execution component checks which actions are currently executing by querying the state management component. It then executes the rules associated with these actions, and in turn propagates the effects of these rules on the entities and actions in the state management

Figure 3: SimSE Graphical User Interface.



Type: SEProject
Description: Roc
RequiredSizeOfC
Budget: 1933000

Type: aCustomer
Name: NASA
Satisfaction: 0.75

I'm on a break now

I'm done creating code

I'm reviewing the requirements document now

Type: Requirements
Name: Requirements
NumKnownErrors: 0
Size: 1.0125

Type: IDE
Name: Eclipse
Cost: 1.0
NumPeopleUsing: 1

Name: Ethan
Energy: 0.400000
Mood: 0.6999999
ProductivityInRe
ProductivityInDe

51 ticks

1

component. After this update is completed, the clock then signals the user interface to update the display. The user interface then queries the state management component and updates itself to reflect the current state.

One of SimSE's most fundamental features is its fully graphical user interface. Learning through visual clues has proven to be far more effective than simply studying textual output [11, 12]. Consider a simulation that wants to teach Brooks' law [7], which states that adding personnel to a project that is late makes it even later than completing it with the original personnel. A text-based simulation environment could only show the effect of this law in numbers that show the project will indeed be later [13]. However, it remains unclear as to *why* the project is later. This is where SimSE will leverage its visual front-end by graphically illustrating that the number of meetings increases and personnel assemble in meeting rooms at a greater frequency. As a result, independent learning is fostered: a student receives direct feedback on their decisions and can draw their own conclusions about the cause-and-effect relationships present in the process.

A preliminary version of the graphical user interface of SimSE is shown in Figure 3. The center part displays a virtual office in which the software engineering process is taking place, including typical office surroundings (e.g., desks, chairs, computers, meeting rooms) and employees. Employees "communicate" with the manager (player) through pop-up "bubbles" that appear over their heads. The player can interact with the employees through right-click menus (not shown) on each employee that display a list of available actions (e.g., do coding, give a raise, fire, ask what they are doing, etc.). Detailed information about each employee (the current values of their attributes) can be obtained by clicking on that employee's image along the bottom edge of the interface. Graphical representations of projects and customers, artifacts (e.g., requirements document, code), and tools are displayed along the top, left, and right edges, respectively. Like the employees, detailed information about each of these entities can be displayed by clicking on its image. The user drives the simulation through the controls of the clock, shown in the lower right corner. They can either step the clock forward a specified number of clock ticks, or step until the next event, i.e., the next time an employee wants to "say" something.

4. Related Work

This research draws from three main areas of related work: simulation in education, software engineering education, and software process simulation. The following subsections describe the relationships between SimSE and these existing bodies of work.

4.1 Simulation in education

Simulation is a powerful educational tool that is widely used in a number of different domains such as flight simulation [14], military training [15], and hardware design [16]. The success of simulation in education can be attributed to its unique qualities that set it apart from other pedagogical approaches: First, simulation allows students to gain valuable hands-on experience of the process being simulated without any of the potential monetary costs or harmful effects that can result from actual real-world experience. As a result, students are free to repeat experiences and experiment with different approaches, their only concern being the *simulated* consequences (rather than real life ones) in case of failure. Second, the relative ease with which simulations are configurable allows the educator to introduce a wide variety of unknown situations for the student to experience. Finally, because simulations operate at a faster pace than real life, students can practice the process many more times than would be feasible in the real world.

4.2 Software engineering education

Software engineering educators have invented new and innovative ways of teaching the subject in recent years. Some of these approaches have included software process simulation designed specifically for education [13, 17, 18]. To date, the most advanced of these is SESAM, a software engineering simulation environment in which students manage a team of virtual employees to complete a virtual project on schedule, within budget, and at or above the required level of quality. The student drives the simulation by typing in textual commands which can consist of hiring and firing employees, assigning them tasks, and asking them about their progress and the state of the project [13]. Although SimSE and SESAM share the purpose of teaching students the software engineering process in a game-based setting, SESAM, unlike SimSE, lacks a visually interesting graphical user interface, which is considered essential to any successful educational simulation [9]. Moreover, despite the fact that the SESAM language is highly flexible and expressive, the model building process is learning- and labor-intensive and requires writing code in a text editor. Despite these drawbacks, SESAM's models do provide a source of some well-documented software engineering rules of behavior that we plan to leverage in our SimSE models.

4.3 Software process simulation

The area of software process simulation deals with creating models of software processes in order to analyze and predict certain properties and behaviors of these processes [19-21]. For example, process simulations have been created to predict the effects of different managerial decisions on the resulting project attributes, such as cycle time, cost, and quality [21]. Because SimSE also simulates a software engineering process, the work done in this area provides useful process models upon which we can base present and future versions of the game. However, since these models were created for the purpose of analysis and discovery rather than education, certain aspects and portions of them are not relevant to the pedagogical goals of the game, and we are adjusting them accordingly as we incorporate them into our models.

5. Conclusions and Future Work

We have presented SimSE, an interactive, graphical, customizable simulation game for software engineering education. SimSE attempts to address the lack of process education present in the typical software engineering course by providing students with a practical, high-level experience of a realistic software engineering process in an engaging manner.

To date, the SimSE model builder and code generator have been completed, and development of the graphical user interface is nearing completion. In parallel, we are designing the explanatory tool and developing two initial simulation models: a high-level model in which an overall software engineering process is simulated and a number of general lessons about the process as a whole are taught, and a second, more detailed model that teaches the roles and regulations of the inspection process by making the student organize and perform a code inspection. In the near future, we plan to evaluate the teaching potential of SimSE by conducting experiments involving undergraduate computer science students at UC Irvine, refine and enhance SimSE based on these results, and continue to build different types of models to demonstrate various phenomena.

6. Acknowledgments

We thank Ethan Lee, Calvin Lee, and Beverly Chan for their contributions to the implementation of SimSE.

Effort partially funded by the National Science Foundation under grant numbers CCR-0093489 and IIS-0205724. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the National Science Foundation. Effort also funded by the UC Irvine CORCLR program.

References:

1. ACM Committee on Computers and Public Policy, *RISKS-FORUM Digest*, 2004: http://catless.ncl.ac.uk/Risks.
2. Jones, C., *Software assessments, benchmarks, and best practices* (Boston, MA: Addison-Wesley, 2000).
3. Callahan, D. and B. Pedigo, Educating Experienced IT Professionals by Addressing Industry's Needs, *IEEE Software*, *19*(5), 2002, p. 57-62.
4. Conn, R., Developing Software Engineers at the C-130J Software Factory, *IEEE Software*, *19*(5), 2002, p. 25-29.
5. McMillan, W.W. and S. Rajaprabhakaran, What Leading Practitioners Say Should Be Emphasized in Students' Software Engineering Projects, *Proceedings of the Twelfth Conference on Software Engineering Education and Training*, 1999, p. 177-185.
6. Shaw, M., Software Engineering Education: A Roadmap, *The Future of Software Engineering*, 2000, p. 373-380.
7. Brooks, F.P., *The mythical man-month: essays on software engineering* (Boston, MA: Addison-Wesley, 1995).
8. Perry, D.E., H.P. Siy, and L.G. Votta, Parallel Changes in Large-Scale Software Development: An Observational Case Study, *ACM Transactions on Software Engineering and Methodology*, *10*(3), 2001, p. 308-337.
9. Ferrari, M., R. Taylor, and K. VanLehn, Adapting Work Simulations for Schools, *The Journal of Educational Computing Research*, *21*(1), 1999, p. 25-53.
10. Beck, K., *Extreme programming explained: embrace change* (Reading, MA: Addison-Wesley, 2000).
11. Higbee, K.L., Recent Research on Visual Mnemonics: Historical Roots and Educational Fruits, *Review of Educational Research*, *49*, 1979, p. 611-629.
12. Shneiderman, B., *Designing the user interface: strategies for effective human-computer interaction* (Boston, MA: Addison-Wesley, 1992).
13. Drappa, A. and J. Ludewig, Simulation in Software Engineering Training, *Proceedings of the 22nd International Conference on Software Engineering*, 2000, p. 199-208.
14. Rolfe, J.M., *Flight simulation (cambridge aerospace series)* (Cambridge, UK: Cambridge University Press, 1988).
15. Lindheim, R. and W. Swartout, Forging a New Simulation Technology at the ICT, *IEEE Computer*, *34*(1), 2001, p. 72-79.
16. Skrien, D., CPU Sim 3.1: A Tool for Simulating Computer Architectures for Computer Organization Classes, *ACM Journal of Educational Resources in Computing*, *1*(4), 2001, p. 46-59.
17. Sharp, H. and P. Hall, An Interactive Multimedia Software House Simulation for Postgraduate Software Engineers, *Proceedings of the 22nd International Conference on Software Engineering*, 2000, p. 688-691.
18. Collofello, J.S., University/Industry Collaboration in Developing a Simulation Based Software Project Management Training Course, *Proceedings of the Thirteenth Conference on Software Engineering Education and Training*, 2000, p. 161-168.
19. Abdel-Hamid, T. and S.E. Madnick, *Software project dynamics: an integrated approach* (Upper Saddle River, NJ: Prentice-Hall, Inc., 1991).
20. Cass, A.G., et al., Little-JIL/Juliette: A Process Definition Language and Interpreter, *Proceedings of the 22nd International Conference on Software Engineering*, Limerick, Ireland, 2000, p. 754-757.
21. Rus, I., J.S. Collofello, and P. Lakey, Software Process Simulation for Reliability Management, *The Journal of Systems and Software*, *46*(3), 1999, p. 178-182.