Teaching Software Engineering Through Simulation

Emily Oh
University of California, Irvine
127B Computer Science Trailer
Irvine, CA 92697-3430
1-949-824-3100
emilyo@ics.uci.edu

ABSTRACT

The common software engineering education method of theory presented in lectures along with application of these theories in an associated class project is insufficient, on its own, to effectively communicate the complex, fundamental dynamics underlying real-world software engineering processes. This paper introduces and lays out plans for SimSE, a detailed, graphical, fully interactive educational software engineering simulation environment that teaches the software process in a practical manner without the time and scope constraints of an actual class project. Once completed, this tool will enable students to form a concrete understanding of the software process by allowing its users to explore different approaches to managing the software process and giving them insight into the complex cause and effect relationships underlying the process.

1. INTRODUCTION

The software engineering industry is still noting a large disparity between the software engineering skills taught at a typical university or college and the skills that are desired of a software engineer by a typical software development organization [5, 11, 13, 16]. This problem seems to stem from the way software engineering is usually taught: a series of concepts and theories are presented in lectures and put into (limited) practice in an associated class project. Although this seems like a reasonable approach at first, lectures and projects alone fail to communicate the following five issues critical to any real world software engineering project: it is non-linear, often encounters multiple, conflicting goals, involves choosing among multiple viable alternatives, involves multiple stakeholders, and may exhibit dramatic consequences.

All of these issues relate to the overall *process* of software engineering. It is difficult to teach this process in lectures due to the fact that students participate passively (by simply sitting and listening to the instructor) while the ideas presented remain abstract. Although class projects involve more active participation by students, time and scope constraints and the dominating focus on deliverables prevent the above issues from being adequately highlighted and communicated to the students. Nonetheless, educating students in these issues is essential to creating a full understanding of the depth and complicated nature of software engineering—preparing them for a future that lies ahead in industry. Essentially, then, what is needed is a way to teach the software process in a practical manner without the drawbacks of an actual class project.

2. SIMULATION GAMES

Simulation/adventure games, such as The Sims [8] and SimCity [7], provide a tremendous source of experience and technology that can successfully be adapted to illustrate the software process. In these games, players strive to fulfill certain—sometimes conflicting—goals by living "virtual lives" in an environment where they are forced to make constant decisions. It is interesting to observe that, in experiencing the consequences of their decisions, players implicitly undergo an experience similar to the software process. In particular, simulation games exhibit strengths in addressing exactly those dimensions that make teaching the software engineering process so difficult:

- **They are non-linear.** Multiple situations that demand the player's attention occur simultaneously, and random factors in the simulation cause every run to be unique.
- **They have multiple, conflicting goals.** Numerous goals that at times interfere with each other must be optimized, generally leading to the attainment of some goals and only the partial fulfillment of others.
- **They allow for the exploration of alternatives.** In case of a wrong decision, a player may quit a game without saving and return to a previously saved state of the game to explore a different alternative.
- **They generally involve multiple stakeholders.** In multi-user games, different players each try to optimize their own results. In single-user games, the game typically provides the stakeholders.
- **They exhibit dramatic consequences.** Decisions made by the player can have drastic effects in the game world, illustrated via graphical depictions.

Aside from sharing these process characteristics with software engineering, simulation in general has proven to be an effective educational technique in several different subject domains (airplane pilot training, hardware design, military training). Simulation facilitates learning through experimentation and "play" in a virtual world where unknown situations are introduced and practiced, alternatives are explored, and experiences are repeated, all in a "safe" environment.

3. HYPOTHESIS

This research project is based on the hypothesis that a game-like, educational software engineering simulation environment is a solution to the problem of adequately teaching the software engineering process. Specifically, simulation games provide an ideal platform upon which to teach software engineering, due to the fact that they exhibit all of the desired characteristics required for teaching the software engineering process, and illustrate many examples of good and effective design that can be utilized to

create an environment that is conducive to learning [12]. Of course, simulation should not replace existing educational techniques, but rather, serve a complementary role. 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.

4. APPROACH

I am in the preliminary stages of addressing the above hypothesis by designing and building SimSE, an educational software engineering simulation environment. The basic architecture of this environment contains three main components: (1) a generic simulation engine, (2) a graphical user interface, and (3) simulation models. Each is discussed next.

4.1 Simulation Engine

The simulation engine is the component that drives the simulation by executing a particular model in a step-by-step fashion. In particular, it takes the current state of the simulation, the model, and any relevant user input, and uses these to calculate the new state of the simulation. It then provides this information to the encompassing simulation environment, which in turn graphically displays the result.

Because the functionality needed by a simulation engine for this environment is similar to that of many existing simulation engines (e.g., SimPack [2], CSIM [1], or SESAM's engine [6]), I intend to evaluate the applicability of these engines and reuse an appropriate solution for my purposes. However, it is likely that the chosen engine will need to be modified, due to two unique requirements of this environment. First, because of its focus on educational use, SimSE requires automated mechanisms to pause, examine, rollback, and continue simulations, encouraging students to fully explore alternative scenarios. Second, in order to promote exploration further, the simulation engine should provide parallel timelines, where each timeline represents a different simulation that evolves independently. Through this feature, students can study the effects of varying certain parameters while keeping other parameters the same, gaining a deeper insight into the cause and effect relationships underlying software engineering phenomena.

4.2 Graphical User Interface

One of the most important features of SimSE is that it will be graphical. Learning through visual clues has proven to be far more effective than simply studying textual output [9, 14]. For example, consider a simulation trying to teach Brooks' Law, which states that adding people to a project that is late will make that project even later, due to the increased need for communication among personnel [3]. Using a text-based simulation environment like SESAM will only reveal the end effect of this law: the numbers show that the project will indeed be later [6]. However, to a student it remains unclear as to *why* the project is later. This is where SimSE will leverage 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 – the student will see personnel assembling in meeting rooms at a greater frequency; it will show complaints from employees that they are unable to get their work done due to these meetings; and it will eventually show that the project is indeed delivered at a later date.

Because it has been shown that a two-dimensional graphical user interface can be as engaging and effective for teaching purposes as a three-dimensional user interface (and due to the time constraint of completing my dissertation), I plan to base SimSE's interface on a tile-based, top-level view of the organization being simulated. Each tile will represent a particular part of the organization, and may contain either stationary artifacts, such as desks, walls, coffee machines, or computers, or a (mobile) participating character, such as a programmer, designer, or project manager. To implement this interface, I plan to utilize one of the several game development kits that are available for building two-dimensional simulation environments [4, 10, 15].

4.3 Models

Because their main purpose is to represent the real-world phenomena that the simulation environment is trying to teach, the models used in driving a simulation are key to any successful learning experience. Thus, the most significant part of this research will involve determining what the requirements are for educationally successful models, and then building a base set of models that meet these requirements. In particular, the models will fall into two classes: small specific models and large generic models. Depending on instructional needs and personal preferences, an instructor will be able to choose a subset of these models for use in a class.

Small specific models provide targeted lessons on specific activities (e.g., "inspections" or "integration testing). These models are designed to develop a student's understanding of the issues involved in carrying out a particular task. For instance, a student going through a simulation of the inspections model has the goal of ensuring that the task is completed on time and follows the standard interaction procedures that typically characterize the inspection process. During the process, they may run into such complications as lack of personnel due to the presence of a major deadline, erratic behavior of employees, or a chaotic meeting that dissolves before the inspection is complete, due to lack of planning and organization on the student's part.

Large generic models introduce an overall view of the software process (e.g., "waterfall model" or "spiral model"), and train students in issues arising in large-scale software processes. These models are at a significantly higher level of complexity than the small models described above. Generally, the objective is for a student to complete one or more software projects in which time, expenditures, and quality must be optimized in parallel. In a simulation of such a model, students will be made aware that certain decisions made early in the process may have effects that only show up after some kind of time delay (e.g., raising expenditures to deliver a requirements document on time may result in a shortage of funds later on at testing time, resulting in a poor quality product that was not adequately tested). Consequently, students will have to continuously make decisions about such questions as whether to raise expenditures to speed up the process or whether to drop the level of quality in an effort to save time.

In order to represent models in such a way that the simulation engine can execute them, they must be specified in a particular, shared modeling language. This language, currently being developed, must model four basic features: the goal to be achieved (e.g., "develop a particular product at minimal cost and a defect rate of less than one percent"), the setting in which the simulation takes place (e.g., the layout of the simulated

organization, its initial set of assets, and available personnel), the underlying rules of the simulation (specifying the behavior of a particular software process, including the available actions, the relations among actions and progress towards achieving certain goals, technological variances, and random effects), and the visual effects to be displayed (e.g., pop-up "bubbles", one or more characters performing an action, personnel moving around).

5. EVALUATION

Once SimSE is built, empirical evaluations will be performed to determine whether the hypothesis underlying the research holds true. In particular, SimSE will be put into use in an introductory software engineering class in order to understand whether its use helps students in achieving a better understanding of the software process. In particular, I will be using three different techniques to evaluate the effectiveness of SimSE. First, the students will be presented with surveys, both during the class and afterward. Second, I will compare the grades of one session of the course, in which students will be introduced to the simulation materials, to the grades of another session, in which students will not be introduced to those materials. Finally, I will track the grades these students earn in subsequent software engineering classes and do the same comparison. If successful, the plan is to package SimSE such that further experiments and evaluations can be performed in collaboration with other universities.

6. CONTRIBUTIONS

This research contributes to both the body of software engineering education and the practice of software engineering in general. Software engineering education will gain a new method of teaching that, when used alongside lectures and projects, is aimed at effectively introducing students to the software process as experienced in real-world software engineering projects. In turn, the practice of software engineering will benefit from receiving graduates that are equipped with this valuable knowledge, and thus, are hopefully better prepared for positions in industry. Furthermore, we believe industrial organizations will also be able to leverage our environment: by using SimSE with models that reflect their organization and processes therein, new employees can be more quickly and successfully trained.

7. REFERENCES

[1] *Mesquite Software Homepage.* 2001. http://www.mesquite.com/htmls/csim18.htm

[2] *SimPack Homepage.* 2001. http://www.simpack.de/

[3] Brooks, F.P., *The Mythical Man-Month: Essays on Software Engineering*. 2 ed. 1995: Addison-Wesley. 336.

[4] Carlin, B., *GameForm.* 2001. http://home.swipnet.se/carlin/gameform

[5] Diaz-Herrera, J.L. and G.M. Powell, *Educating Industrial-strength Software Engineers*, in *Proceedings of the Eleventh Conference on Software Engineering Education and Training*. 1998, IEEE Computer Society. p. 139-150.

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

[7] Electronic Arts, *SimCity 3000*. 1998.

[8] Electronic Arts, *The Sims*. 2000.

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

[10] Marty, B., *The Scrolling Game Development Kit.* 2001. http://gamedev.sourceforge.net/

[11] McMillan, W.W. and S. Rajaprabhakaran, *What Leading Practitioners Say Should Be Emphasized in Students' Software Engineering Projects*, in *Proceedings of the Twelfth Conference on Software Engineering Education and Training*, H. Saiedian, Editor. 1999, IEEE Computer Society. p. 177-185.

[12] Randel, J.M., et al., *The Effectiveness of Games for Educational Purposes: A Review of Recent Research.* Simulation and Gaming, 1992. 23(3): p. 261-276.

[13] Shaw, M., *Software Engineering Education: A Roadmap*, in *The Future of Software Engineering*, A. Finkelstein, Editor. 2000, ACM. p. 373-380.

[14] Shneiderman, B., *Designing the User Interace: Strategies for Effective Human-Computer Interaction*. 2nd ed. 1992: Addison-Wesley Publishing Company.

[15] Wiering, M., *TileStudio.* 2001. http://www.cs.kun.nl/is/ts/

[16] Wohlin, C. and B. Regnell, *Achieving Industrial Relevance in Software Engineering Education*, in *Proceedings of the Twelfth Conference on Software Engineering Education and Training*, H. Saiedian, Editor. 1999, IEEE Computer Society. p. 16-25.