Teaching Software Engineering through Simulation

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

A large difference exists 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. At the heart of this difference seems to be the way software engineering is typically introduced to students: general theory is presented in a series of lectures and put into (limited) practice in an associated class project. While at first this seems to be a reasonable approach, practical, didactic, and timing reasons necessarily lead to the fact that such lectures and class projects often lack an in-depth treatment of the following five issues critical to any real-world software engineering project:

- *Software engineering is non-linear.*
- *Software engineering often has multiple, conflicting goals.*
- *Software engineering continuously involves choosing among multiple viable alternatives.*
- *Software engineering involves multiple stakeholders.*
- *Software engineering may exhibit dramatic consequences.*

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

We believe that simulation/adventure games, such as SimCity, The Sims, Escape from Monkey Island, Myst, and Ultima Online, provide a tremendous amount of experience that can be leveraged to effectively illustrate the software process in the way that software engineering education today is lacking. In these games, players work towards achieving certain, sometimes conflicting goals, by living their “virtual lives” in such a way that they must make tradeoffs in choosing to work towards certain goals while ignoring others, much like the process of software engineering. Specifically, the player’s experience in these games exhibits all of the difficult-to-teach, yet critical characteristics of the software engineering process:

- **They are non-linear.** Multiple events happen at the same time; one must 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 involve multiple, conflicting goals.** As explained previously, 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 certain goals that are attained and others that can only be partially fulfilled.
- **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 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-player games, the game simulation provides the stakeholders. For example, SimCity has unions and Green Party representatives that the player must keep happy in making decisions regarding city planning.
- **They exhibit dramatic consequences.** Although not real, the graphical illustration of these dramatic consequences (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.

In addition to sharing these process characteristics with software engineering, simulation has also proven to be an effective educational technique in other domains, including airplane pilot training, military training, and hardware design. In each of these cases, simulation provides significant educational benefits: valuable experience is accumulated without the potential of the dramatic consequences that may occur in case of failure. Moreover, unknown situations can be introduced and practiced, experience can be repeated, alternatives can be explored, and a general freedom of experimentation and "play" is promoted in the training exercise.

We are currently designing and building an educational software engineering simulation environment that adopts simulation game technology to teach students the software process in accordance with the above objectives. We are in the early stages of our project, and believe the workshop can provide valuable input and feedback to the project—hence our desire to attend.

About the Authors

Emily Oh is a Ph.D. student in the Department of Information and Computer Science at the University of California, Irvine. Her area of specialty is software engineering education, and she has published two papers on the research outlined here.

André van der Hoek is an assistant professor in the Department of Information and Computer Science at the University of California, Irvine. His specialty is software engineering and one of his continuing interests is to find novel ways to educate students in software engineering.