On the Role of Learning Theories in Furthering Software Engineering Education

Emily Oh Navarro and André van der Hoek
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

Learning theories describe how people learn. There is a large body of work concerning learning theories on which to draw, a valuable resource of which the domain of software engineering educational research has thus far not taken full advantage. In this chapter, we explore what role learning theories could play in software engineering education. We propose that learning theories can move the field of software engineering education forward by helping us to categorize, design, evaluate, and communicate about software engineering educational approaches. We demonstrate this by: (1) surveying a set of relevant learning theories, (2) presenting a categorization of common software engineering educational approaches in terms of learning theories, and (3) using one such approach (SimSE) as a case study to explore how learning theories can be used to improve existing approaches, design new approaches, and structure and guide the evaluation of an approach.

Keywords: software engineering education, educational evaluation, simulation, educational technology, instructional technology, instructional evaluation, learning theories, educational games

Introduction

Learning theories are attempts to describe and understand the various ways in which people learn. They are an important resource for educational research, as they can both guide us in creating new educational approaches, and help us analyze and improve existing approaches.

In this book chapter, we propose that learning theories, which have thus far been explicitly leveraged in software engineering education in only a minimal way, can actually play quite a significant role in this domain. Specifically, we believe that learning theories can serve to move the field of software engineering education forward by helping us to categorize, design, evaluate, and communicate about software engineering educational approaches. Categorizing approaches in terms of learning theories can help us to understand the approaches in relation to each other, understand how they fit together, and point out areas of untapped potential. New approaches can be designed to leverage certain theories whose potential is unfulfilled or known to be especially valuable in our domain. Learning theories can be used to evaluate approaches by helping structure experiments to look for the presence of these and other theories in the processes of learners. And, we can use our newfound knowledge to communicate in a common language—that of learning theories—about different approaches and our experience with them.

This chapter details this vision of principally using learning theories in the domain of software engineering education. We first briefly present a set of well-known (mainly constructivist) learning theories that are especially applicable. We then introduce a categorization of the major software engineering educational approaches to date in terms of the learning theories that they appear to have been designed around. Following this, we discuss the role

learning theories can play in analyzing and improving the design of a software engineering educational approach (and designing new approaches), and focus on the analysis of one such approach (SimSE) as a case study. We then discuss how software engineering educational approaches can be evaluated in terms of learning theories, again using SimSE as a case study. We conclude with a summary in the final section.

Background - Learning Theories

To provide some background for our discussion on the role of learning theories in software engineering education, in this section we will briefly introduce the set of learning theories that we surveyed for the purposes of our analysis. We do not include here an exhaustive list of all learning theories with significant detail. Instead, the purpose of this section is to simply introduce some of the ones we have seen software engineering educational approaches centered around most frequently, and provide pointers to where more information about each one can be found. In addition, we will also briefly touch on implications and typical or possible applications of each theory for software engineering education.

We chose the particular set of learning theories discussed here because of two criteria: relevancy to software engineering and orthogonality among the factors defining the theory. In other words, these theories are the ones we have seen to be most clearly and/or frequently embodied in the software engineering educational approaches that we surveyed. Furthermore, there exists a great deal of overlap among learning theories, and there are several learning theories that encompass a number of others. In these cases, we either group theories that have the same basic idea, and omit those that simply combine a number of theories.

We acknowledge that these theories fall mainly into the constructivist paradigm (rather than the behaviorist or cognitive categories), however, given that constructivism is the most recently-developed paradigm, and software engineering is a relatively new discipline, this is not surprising (it has been argued elsewhere, in fact, that the evolution of computer science education in the past decade or so has been significantly influenced by constructivism (Kolikant, 2001)). While it is certainly true that most delivery methods generally contain a mix of various theories that fall into each of the three camps (constructivist, behaviorist, and cognitive), because the constructivist aspects are the most focused on, we have chosen to scope this survey and analysis to focus primarily on these theories. Surely similar surveys and analyses could be done with cognitive and behaviorist theories that would yield interesting results, however, such exercises are outside the scope of the one presented here. Nevertheless, some of the theories surveyed in this chapter do have elements of cognitive and/or behaviorist principles. For example, Learning through Failure involves a form of "punishment" (failure) meant to "extinguish" a certain behavior.

An additional issue that should be noted is the distinction between learning "theories," learning "models," and learning "methods," as well as their counterparts in the domain of instructional design (instructional design theories, models, and methods). Because the lines between these are blurred and often used interchangeably, it should be noted that in this paper several of the "learning theories" we refer to can also be called by some of these other terms. When this is the case, we will point it out in our discussion of those theories. However, as is frequently done in the literature, we use the term "learning theory" broadly, as a term that covers all of these categories.

One of best-known learning theories is *Learning by Doing*, a theory based upon the premise that people learn a task best not by hearing about it, but by actually *doing* it (Dewey, 1916). The

implication of this theory for instructional design is the following: the learner should be provided with ample opportunity to actually perform the activities they are meant to learn, rather than using passive mediums such as lectures and readings. In software engineering education, this translates to going beyond just lectures and reading assignments (although, for most any domain, a certain amount of such scaffolding is necessary to provide the learner with the required background knowledge to effectively participate in the learning by doing). Software engineering educators have recognized this, and now a standard component of nearly all software engineering courses is the class project—a small software engineering project that students must develop using some of the techniques learned in class.

Situated Learning (Lave, 1988) is an educational theory that builds upon the Learning by Doing approach. While Learning by Doing focuses on the specific learning activities that the student performs, the Situated Learning theory is concerned with the environment in which the learning by doing takes place. In particular, Situated Learning is based on the belief that knowledge is situated, being in large part a product of the activity, context, and culture in which it is developed and used. Therefore, the environment in which the student practices their newly learned knowledge should be "authentic", resembling, as closely as possible, the environment in which the knowledge will be used in real life. A popular application of this theory in software engineering education focuses on incorporating aspects of realism (or "authenticity") into the class project, such as using an industrial participant to play the role of the customer (Hayes, 2002), using maintenance- or evolution-based projects (McKim & Ellis, 2004), or using large teams of people that are distributed across geographical locations (Favela & Pena-Mora, 2001).

Like Situated Learning, *Keller's ARCS Motivation Theory* (Keller, 1983) also focuses on motivating students to learn. However, rather than focusing on the physical environment in which they learn, Keller's ARCS Motivation Theory concerns itself with producing certain feelings in the learner that are believed to promote learning. In particular, these feelings are attention, relevance, confidence, and satisfaction.

- **Attention:** The attention and interest of the learner must be engaged. Proposed methods for doing so are: introducing unique and unexpected events; varying aspects of instruction; and arousing information-seeking behavior by having the learner solve or generate questions or problems.
- **Relevance:** Learners must feel that the knowledge is relevant to their lives. The theory suggests that knowledge be presented and practiced using examples and concepts that are relevant to learners' past, present, and future experiences.
- **Confidence:** Learners need to feel personal confidence in the learning material. This should be done by presenting a non-trivial challenge and enabling them to succeed at it, communicating positive expectations, and providing constructive feedback.
- **Satisfaction:** A feeling of satisfaction must be promoted in the learning experience. This can be done by providing students with opportunities to practice their newly learned knowledge or skills in a real or simulated setting, and providing positive reinforcements for success.

Keller's ARCS is technically considered an instructional design model that is rooted in various learning theories. Two of the most directly contributing theories are Andragogy (Knowles, 1984) and Expectancy-Value Theory (Fishbein & Ajzen, 1975). Andragogy concerns adult learners in particular, and focuses on their need for self-directed, relevant, hands-on learning. Expectancy-Value Theory states that in order for a learner to put forth the effort required to learn, they must both value the knowledge/task/exercise and expect that they can

succeed at it. Because Keller's ARCS combines these theories and provides more hands-on applicability than either theory alone, we have chosen to include it (rather than the theories it is based on) in our survey and analysis.

While Keller's ARCS could be applied in a number of different ways in software engineering education, in general it entails providing the students with attention-grabbing, realistic, hands-on assignments that pose a significant, yet doable challenge. One class of approaches that explicitly sets out to accomplish such goals is that in which the class project is made purposely open-ended and/or vague. This is done in two main ways: either by allowing the students to define their own requirements (giving students the pseudo-experience of new product development based on market research) (Navarro & van der Hoek, 2005b), or by allowing them to define their own process (giving students experience in not only following a process, but in designing the process that they follow) (Groth & Robertson, 2001). The stated purpose of these open-ended approaches is to mimic common, less-structured (authentic) real-world software engineering situations, giving the students more ownership of the project and therefore more interest in it, as well as a greater feeling of confidence and satisfaction when the project is completed.

Model-Centered Instruction (Gibbons, 2001) (which is also considered an instructional design theory) says educators should center all learning activities around models of three types: models of environments, models of cause-effect systems, and models of human performance. Presentation of general concepts and theories should be kept to a minimum. Instead, Model-Centered Instruction believes that knowledge is best learned by exploration of these models. In software engineering education, this translates to simulating realistic situations, presenting case studies, and assigning realistic problems for the students to solve. One software engineering educational approach that embodies this theory is the practice-driven one, in which the curriculum is largely lab- and project-based, and lectures are used only as supporting activities (Ohlsson & Johansson, 1995).

The *Discovery Learning* theory (Bruner, 1967) takes a similar approach to Model-Centered Instruction in that it believes that an exploratory style of learning is best. Discovery Learning is based on the idea that an individual learns a piece of knowledge most effectively if they discover it on their own, rather than having it explicitly told to them. This theory encourages educational approaches that are rich in exploring, experimenting, doing research, asking questions, and seeking answers. Educational software engineering simulation approaches (Drappa & Ludewig, 2000; Navarro & van der Hoek, 2005a) are specifically designed to facilitate this type of learning—no knowledge is made explicit in the simulation, as it is rather discovered by students experimenting with different approaches and seeing the effects of their decisions on the outcome of the simulation. These types of approaches are generally given as structured exercises and combined with other teaching methods (such as lectures, readings, and projects). Including this type of scaffolding has been found to be crucial in making Discovery Learning maximally effective (Kirschner *et al.*, 2006; Roblyer, 2005).

Along the same lines as the Discovery Learning theory is the *Learning Through Failure* theory (Schank, 1997). This theory is based on the assumption that the most memorable lessons are those that are learned as a result of failure. The theory argues that: (1) Learning through failure provides more motivation for students to learn, so as to avoid the adverse consequences that they experience firsthand when they do not perform as taught, and (2) Failure engages students, as they are motivated to try again in order to succeed. Proponents of the theory argue that students should be allowed to (and even set up to) fail to encourage maximal learning. Although Learning through Failure is usually applied to the realm of e-learning, there have also

been some non-e-learning software engineering educational approaches in which the main avenue of learning is through failure. In these "sabotage" approaches, the instructor purposely sets the students up for failure by introducing common real-world complications into projects (e.g., crashing hardware just before a deadline), the rationale being that students will then be prepared when these situations occur in their future careers (Dawson, 2000).

The theory of *Learning through Reflection* is primarily based on Donald Schön's work suggesting the importance of reflection activities in the learning process (Schön, 1987). In particular, Learning through Reflection emphasizes the need for students to reflect on their learning experience in order to make the learning material more explicit, concrete, and memorable. Some common reflection activities include discussions, journaling, or dialogue with an instructor (Kolb, 1984). One example of this in software engineering is (Tomayko, 1996), a practice-driven industrial partnership approach that incorporates weekly one-on-one mentoring sessions with a "coach" to discuss each student's performance and help them reflect on their experience. The game-based simulation described in (Drappa & Ludewig, 2000) and the industrial simulation described in (Nulden & Scheepers, 2000) also incorporate dialogue and reflection as post-simulation activities in which students analyze and discuss their simulation experience with a tutor or instructor, and reflect on what they have learned.

Finally, the theory of *Elaboration* (Reigeluth & Rodgers, 1980) states that, for optimal learning, instruction should be organized in order of complexity, from least complex to most complex. Simplest versions of tasks should be taught first, followed by more complicated versions. This is a theory that is generally inherent to most curricula (as well as most other learning theories), as courses and topics are usually introduced in order of increasing complexity. In software engineering educational approaches, applying this theory can sometimes be difficult, as there is oftentimes no natural way to organize the information in terms of complexity (e.g., how can one do this for a class project?). One approach that has been able to do this is the industrial simulation approach described in (Collofello, 2000). In this approach, students are assigned very simple simulations to begin with, and the complexity of the simulations is incrementally increased as the students progress in their knowledge.

As mentioned previously, what has been presented in this section is only a brief introduction to the relevant learning theories. There is much more detail to these theories than what we have discussed, detail which must be looked into further before one can effectively apply these theories to their educational approaches. Typically, subtleties are involved in each one, and care must be taken to pay attention to these details.

Learning Theory-based Categorization of Existing Approaches

One of the main ways that learning theories can be used in software engineering educational research is to provide the field with a way to analyze and categorize existing approaches, both independently and in relation to each other. Such a categorization can serve to help us understand how the different approaches fit together and create a picture of the field as a whole, so that areas of strengths, weaknesses, and untapped potentials can be unearthed. We have done such a categorization, which we will present in this section.

Before creating this categorization, in order to organize our analysis we first surveyed the major software engineering educational approaches published in the past several years and found that they can be lumped into three broad groupings: *realism*, *topical*, and *simulation* (these groupings can be broken down further into sub-groupings, as shown in Table 1). Realism approaches are those that focus on making various aspects of the students' project experience more closely resemble one they would encounter in the real world. Some of these have included industry participation (Beckman *et al.*, 1997; Kornecki *et al.*, 2003; Wohlin & Regnell, 1999), emphasizing non-technical skills such as marketing and project management (Gnatz *et al.*, 2003; Goold & Horan, 2002), and focusing on making the nature and composition of the student teams that work on the project more realistic (e.g., making them very large (Blake, 2003) or composed of several sub-teams (Navarro & van der Hoek, 2005b)). Topical approaches aim to educate students in detail about a topic generally not covered in depth in mainstream textbooks and lectures. These approaches do not focus on specific delivery methods, but instead focus on the mere addition of the topic as a crucial component of an effective and complete education in software engineering. Some examples of such topics are formal methods (Abernethy & Kelly, 2000), real-time software engineering (Kornecki, 2000), and specific software processes such as the Personal Software Process (Hilburn, 1999) or the Rational Unified Process (Halling *et al.*, 2002). Finally, simulation approaches are those that have students practice software engineering processes in a (usually) computer-based simulated environment. Within the realm of software engineering simulations, there are three main types: industrial simulations brought to the classroom (Collofello, 2000; Pfahl *et al.*, 2000), game-based simulations (Drappa & Ludewig, 2000; Navarro & van der Hoek, 2005a), and group process simulations (Nulden & Scheepers, 2000; Stevens, 1989).

Table 1: Grouping of Software Engineering Educational Approaches.

Realism	**53**	**Topical**	**48**	**Simulation**	**8**
Industrial Partnerships	**16**	**Formality**	**3**	**Industrial**	**2**
- Modify real software	1	- Formal methods	2	**Game-Based**	**4**
- Industrial advisor	1	- Engineering	1	**Group Process**	**2**
- Industrial mentor/lecturer	2	**Process (Specific)**	**21**		
- Case study	5	- PSP	14		
- Real project / customer	7	- TSP	2		
Maintenance/Evolution	**9**	- RUP	3		
- Multi-semester	4	- XP	2		
- Single-semester	5	**Process (General)**	**6**		
Team Composition	**13**	- Process engineering	3		
- Long-term teams	1	- Project management	3		
- Large teams	3	**Parts of Process**	**3**		
- Different C.S. classes	1	- Scenario-based req. eng.	1		
- Different majors	2	- Code reviews	1		
- Different universities	2	- Usability testing	1		
- Different countries	1	**Types of Software Eng.**	**8**		
- Team structure	3	- Maintenance/Evolution	3		
Non-Technical Skills	**2**	- Component-based SE	2		
Open-Endedness	**7**	- Real-time SE	3		
- Requirements	2	**Non-Technical Skills**	**7**		
- Process	5	- Social/logistical skills	3		
Practice-Driven	**3**	- Interact w/ stakeholders	1		
Sabotage	**3**	- HCI	2		
		- Business aspects	1		

To categorize these approaches in terms of learning theories, we carefully studied each one to determine which learning theories appear to have been applied (whether intentionally or unintentionally), and which learning theories have clear potential to be employed. The resulting categorization is presented in Table 2 as a matrix of approaches and the learning theories that they leverage. (For a complete discussion of this categorization, see (Navarro, 2005)—here we present only the highlights.) The presence of three stars in the table indicates that the approach embodies the particular theory, or is centered around it. The presence of two stars represents that the theory appears to be involved in the design of that type of approach, but is perhaps not an intrinsic part of it, and may not be involved in all approaches that fall within that type. The presence of one star indicates that there is an obvious potential for that particular type of approach to employ that learning theory, but there have been very few, or no known cases of it.

Table 2: Software Engineering Educational Approaches and the Learning Theories they Incorporate.

	Learning by Doing	Situated Learning	Keller's ARCS	Model-Based Instruction	Discovery Learning	Learning Through Failure	Learning Through Reflection	Elaboration
Industrial Partnership – Real Project	**	***	**				*	
Maintenance/Evolution	**	***					*	**
Team Composition	**	***					*	
Open-Endedness	**	**	***		**	**	*	
Non-Technical Skills	**	**					*	
Practice-Driven	***			***	***	**	*	*
Sabotage	**	**				***	*	
Topical	**	*	*	*	*	*	*	*
Simulation	***	**	***	*	***	**	*	**

Example: Simulation and Learning Theories

As an example of how we analyzed each approach in terms of learning theories, in this section we will focus on the simulation category and walk through how we determined the applicability of each learning theory for these approaches. First of all, all aforementioned educational software engineering simulations allow students to learn software processes by participating in them (Learning by Doing), albeit virtually. This theory is central to the paradigm of educational simulations (hence, the three stars in the table). These simulations also employ Situated Learning by adding realism to the learning environment, although in different ways: Industrial simulations

add realistic factors in the form of real project data in the simulation model; Game-based simulations add realism by immersing the student in the role of a participant in a realistic game scenario; Group process simulations inject realism through the simulated characters that behave similarly to real-world participants. Because these realistic factors are artificial in that they are virtual (rather than in a real-life setting), we put two stars in the table for this theory.

Simulation approaches strongly fit with the Keller's ARCS model of learning. In particular, they are specifically designed to promote attention, relevance, confidence, and satisfaction (and have been shown to do so in some cases) in the following ways:

- **Attention:** A number of studies done with educational software engineering simulations have repeatedly shown that students find these simulations enjoyable, engaging, and an interesting challenge they are happy to take on (Baker *et al.*, 2003; Dantas *et al.*, 2004; Navarro & van der Hoek, 2005a; Sharp & Hall, 2000; Stevens, 1989). This is particularly true for game-based simulations. Clearly this is the result of the elements of surprise, humor, challenge, and fun that are integral to many game-based simulations.
- **Relevance:** Because learners can experience firsthand how the knowledge they are learning is relevant in a real-world situation (the one that is portrayed in the simulation), simulation promotes a feeling of relevance to students' future careers. This relevance can be enhanced by the usage of real-world data in the model to make the simulation more realistic. Furthermore, as the theory suggests, relevance is enhanced even further if the educational approach builds on previous and present knowledge. Simulations that are used to demonstrate concepts that have already been communicated to the students in another form (e.g., lecture or text) directly address this.
- **Confidence:** Simulations provide a non-trivial challenge that is also doable. As students are given the opportunity to succeed at a simulation, they will feel a sense of personal confidence in the learning material. This is especially true in game-based simulations, in which students have the additional benefit of feeling they have "won the game."
- **Satisfaction:** As students are able to practice their knowledge and skills in a realistic (yet simulated) setting, seeing the positive consequences of applying their knowledge correctly promotes a true feeling of satisfaction. Again, game-based simulations add to this if the student is also rewarded with a high score or some other game-relevant measure of success.

Model-Based Instruction has not been utilized at all in simulation, but has obvious potential to be. In particular, simulations could be used as the model (realistic situation, case study, *and* problem, simultaneously) that instruction is centered around. In such a case, students would practice a simulation (or series of simulations) for each concept (or set of concepts) being taught. Simulations would allow for ample exploration—one of the basic tenets of Model-Based Instruction—as students could practice the same simulation multiple times, using a different approach each time, learning the consequences of various actions, and, as a result, learning a great deal about the process and concepts being simulated.

The exploratory quality of simulation in and of itself directly implements the Discovery Learning theory. The nature of simulation is highly conducive to allowing students to discover knowledge on their own, as they see phenomena played out in a simulation, and are encouraged to explore, experiment, do research, ask questions, and seek answers.

This type of exploratory learning is also inherently related to the Learning through Failure theory. As students explore the simulation and try different approaches, they are likely to fail at least a few times. In fact, one of the basic purposes of simulations is to allow students to "push

boundaries", try different approaches, and fail without fear of the drastic and severe consequences that would occur in a real-world setting. For example, a student who fails in a simulated software project would only have to worry about getting a low game score or seeing an unhappy simulated customer, while in the real world such a failure could cost millions of dollars or have even more serious consequences.

Learning through Reflection has also been incorporated into simulation approach, although only limitedly: with the game-based simulation SESAM (Drappa & Ludewig, 2000), and the industrial simulation described in (Nulden & Scheepers, 2000). As mentioned previously, dialogue and reflection sessions have been incorporated into these learning processes as post-simulation activities. Some dialogue activity is also an inherent part of Problems and Programmers (Baker et al., 2003), the educational software engineering card game simulation. The face-to-face, competitive nature of this physical card game has been shown to promote rich and useful discussion between student opponents, regarding such topics as why they took the approach they did, the reasons behind one person's win and another's loss, and their reactions to unexpected events.

Finally, the Elaboration theory has also been only limitedly incorporated into simulation-based software engineering educational approaches. In particular, Elaboration has only been leveraged in the process used with the industrial simulation described in (Collofello, 2000). This process consists of assigning students very simple simulations to begin with, and incrementally increasing the complexity of the simulations as the students progress in their knowledge.

Categorization Highlights

The first thing to notice in general from Table 2 is that, although learning theories are not often explicitly discussed in software engineering education research, they are indeed applicable in our domain. Whether consciously or unconsciously, people have been building approaches toward them in various ways. If we look at how the different learning theories fare with respect to the number of approaches that incorporate them, we can clearly see that our domain has focused the most on Learning by Doing and Situated Learning. This is not a surprise, given the strong emphasis on preparing students for the "real world" that is intrinsic to the field. In contrast, Learning through Reflection is the most under-explored theory, but also has the most potential for greater use—every category of approach has the potential to leverage (or better leverage) this theory.

If we then look at each approach with respect to the learning theories they incorporate, we can see that most of them apply multiple theories at once. The "topical" category has one star for each theory because, since these approaches focus on the topic rather than on delivery methods, they theoretically have the potential to apply all of the theories, depending on the way that topic is taught. Simulation, on the other hand, directly incorporates, or has the potential to directly incorporate all of the theories considered in some way or another. While it certainly is not the case that any teaching method that addresses more learning theories than another is better than that other method (consider a combination of strategies put together haphazardly in some teaching method versus one well-thought-out and tightly-focused method cleverly leveraging one very good strategy), an approach that naturally addresses factors and considerations of multiple learning theories is one that is most definitely worth exploring. Simulation is such an approach, but one that has been significantly underexplored in software engineering education (Navarro, 2005)—something that we are attempting to address with the approach described in the following section.

Detailed Analysis/Design/Development of an Approach in terms of Learning Theories

In addition to providing the field with a way to categorize and analyze existing software engineering educational approaches, learning theories can also help in developing new approaches and modifying existing approaches to be more effective. Categorizations such as the one presented in the previous section can help guide the design (or re-design) of such approaches, as areas for potential are highlighted.

Case Study: The Design of SimSE

In this section, we present a case study of a software engineering educational approach that was actually not explicitly designed with learning theories in mind. In looking back at our approach in light of learning theories, however, we can see that several of our key decisions made in its design are highly relevant to some of these theories. We can also see missed opportunities of ways we could have leveraged additional learning theories to make it more effective.

The approach is SimSE, an educational game-based software engineering simulation environment. SimSE is a computer-based environment that facilitates the creation and simulation of realistic software process simulation models—models that involve real-world components not present in typical class projects, such as large teams of people, large-scale projects, critical decision-making, personnel issues, multiple stakeholders, budgets, planning, and random, unexpected events. In so doing, it aims to provide students with a platform through which they can experience many different aspects of the software process in a practical manner without the overarching emphasis on creating deliverables that is inherent in actual software development.

The graphical user interface of SimSE is shown in Figure 1. 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 an assigned software engineering project or task. The player drives the process by, among other things, hiring and firing employees, assigning tasks, monitoring progress, and purchasing tools. At the end of the game, the player receives a score indicating how well they performed, and an explanatory tool provides them with a visual analysis of their game, including which rules were triggered when, a trace of events, and the "health" of various attributes (e.g., correctness of the code) over time (See Figure 2).

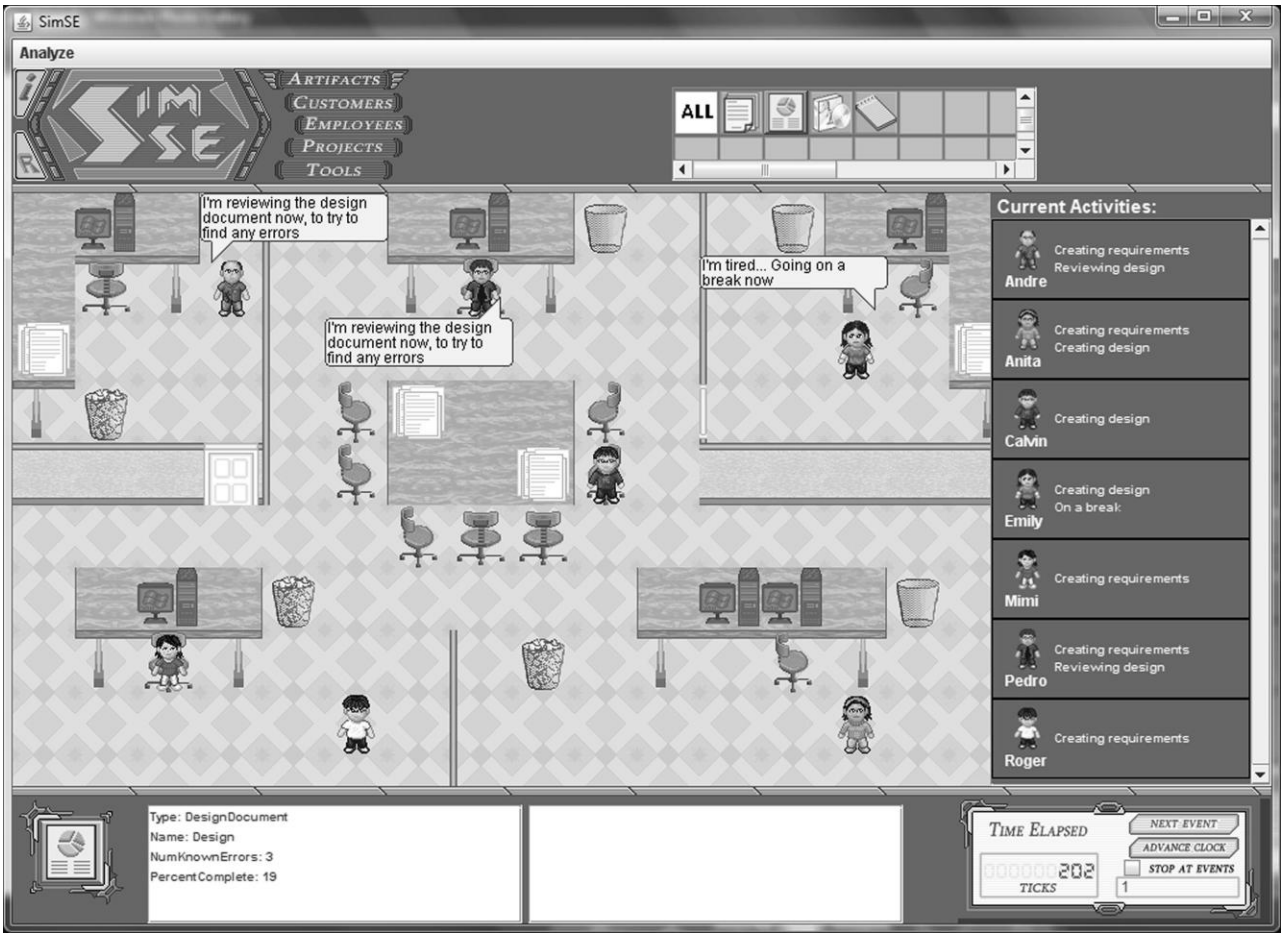

Figure 1: SimSE Graphical User Interface

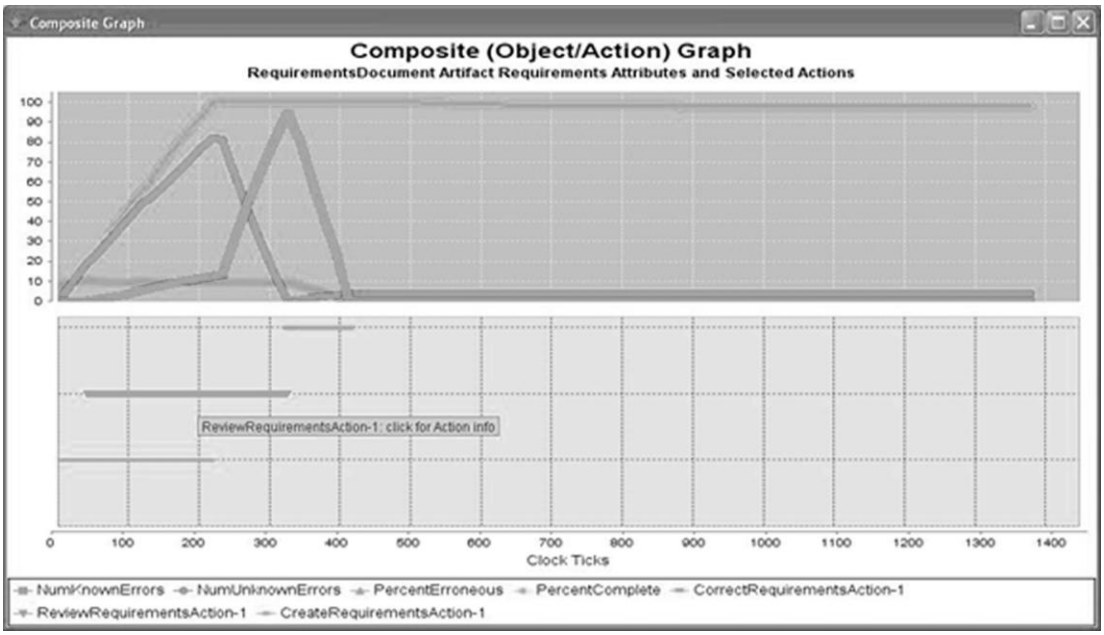

Figure 2: Graphical Representation of a SimSE Game, Generated by the Explanatory Tool.

To date, six SimSE game models exist: a waterfall model, an inspection model, an incremental model, an Extreme Programming model, a rapid prototyping model, and a Rational Unified Process model. For more information on SimSE, including its design, game play, and simulation models, see (Navarro, 2006).

The idea of SimSE was originally motivated by the hypothesis that simulation can bring to software engineering education many of the same benefits it has brought to other educational domains. Specifically, we believed that software engineering *process* education could be improved by using simulation to allow students to practice managing different kinds of "realistic" software engineering processes. The constraints of the academic environment prevent students from having the opportunity to practice many issues surrounding the software engineering process in their course projects. Our approach therefore focused on providing this opportunity through the use of simulation.

To guide us in the design of SimSE, we performed two activities: (1) a study of the domain of software engineering education to discover what its unique needs are, and (2) a survey of well-known principles for successful educational simulations from the research literature. The result of this was a specific set of key decisions that are listed here and discussed in light of the learning theory (or theories) that we later discovered related directly to them:

1. **Use of the Game Paradigm.** We could have chosen to base our simulation approach on the industrial simulation or group process simulation paradigms mentioned previously, but instead we chose the game paradigm. It has been shown that game-like features such as graphics, interactivity, surprising random events, and interesting, life-like challenges are known to hold a student's attention and promote a feeling of confidence and satisfaction as they succeed in the game (Ferrari *et al.*, 1999). This directly corresponds to the Keller's ARCS theory, which suggests that such qualities promote a highly effective learning experience.
2. **A Fully-Graphical User Interface.** To make SimSE maximally engaging and visually realistic, we chose to design a fully graphical, rather than textual interface. As was shown in Figure 1, the focal point of this interface is a typical office layout in which the simulated process is "taking place", including cubicles, desks, chairs, computers, and employees who "talk" to the player through pop-up speech bubbles over their heads. In addition, graphical representations of all artifacts, tools, customers, and projects along with the status of each of these objects are visible. This decision to graphically portray simulated software engineering situations turned out to be strongly in line with the theory of Situated Learning—the learner is provided with a visual context that corresponds to the real world situations in which the learned knowledge would typically be used**.**
3. **A High Level of Interactivity.** Keeping the attention of the learner engaged is not only done by making a user interface visually appealing, but also by continuously involving the learner. Thus, rather than designing SimSE as a continuous simulation that simply takes an initial set of inputs and produces some predictive results, we designed it in such a way that the player must make decisions and steer the simulation accordingly throughout the entire process. SimSE operates on a step-by-step, clock tick basis, and every clock tick the player has the opportunity to perform actions that affect the simulation. Keeping the learner continuously engaged and giving them ample opportunity to practice their skills and tackle challenges are tactics suggested by the Keller's ARCS theory for promoting attention, relevance, confidence, and satisfaction.

4. **Customizable Simulation Models.** SimSE includes a model builder tool and associated modeling approach that allow an instructor to build simulation models and generate customized games based on these models. This feature adds the (unanticipated) potential for using SimSE in a way that follows the theory of Elaboration—instructors could build models of varying complexity and use them in order of increasing complexity with students. Although we have not yet built such models with SimSE, it is in our future plans to do so, as we now know that this potential for greater effectiveness is there.
5. **An Explanatory Tool.** An integral part of SimSE is its novel explanatory tool that provides players with a visual representation of how the simulated process progressed over time and explanations of the rules underlying the game. This feature promotes Learning through Reflection as it allows players to look back on their game and analyze their decisions and how those decisions affected the outcome. The explanatory tool output could also potentially be used as the focal point of a dialogue session between student and tutor/instructor.
6. **Complementary Usage of SimSE.** Rather than design SimSE to be a standalone tool meant to replace standard course components such as lectures, readings, and projects, we instead designed it to be used complementary to them, and have used it in such a setting. The simulation models we have built require a basic set of knowledge and skills in order to play and learn from them effectively, knowledge that students conceivably obtain in lectures and readings. Thus, in essence, SimSE allows them to "Learn by Doing" by learning through experience the lessons communicated through reading and lectures, as well as other lessons that are simply not adequately teachable through passive means. Linking the knowledge learned in SimSE to existing knowledge also promotes the feeling that what a student is learning is of relevance to them, a major tenet of Keller's ARCS.
7. **Simulation models that provide a clear goal.** SimSE allows the modeler to compose a "starting narrative" for the player that appears at the start of a game, and to which the player can refer back at any time during a game. In the models we have built, we have used this starting narrative to provide the player with the exact goals of the simulation, criteria for completion of these goals, and any hints or special notes that might help them along the way. Precisely defined objectives not only guide students through a simulation, but also pose a challenge that many students find hard to resist. Achieving the goal becomes a priority and Discovery Learning is employed as creative thinking is sparked in coming to an approach that eventually achieves that goal.
8. **Simulation models that are adequately challenging.** We have built into our simulation models interesting situations that are adequately challenging (engaging students' attention and making it likely that they learn through failure at times) but not impossible, promoting eventual success that leads to confidence in the learning material and satisfaction in the experience (central principles to Keller's ARCS).

Looking back on the design of SimSE in light of learning theories served to link some of our intuition in the design of SimSE to these theories, thereby increasing our confidence of being on the right path with our approach. In addition to this, it also revealed some missed opportunities that we could have taken advantage of, had we originally designed SimSE with learning theories in mind. For example, we could have better taken advantage of the Elaboration theory by designing our models in incrementally complex versions, and introducing them to students in order of increasing complexity. In our usage of SimSE in courses and in out-of-class studies, we also could have made reflection a more central and structured part of the approach by providing

the student with explicit explanatory tool exercises to complete, exercises that would encourage the type of reflection that would help solidify the lessons learned in the simulation (currently, the student is simply given the explanatory tool, and decisions about how to use it are left up to them). As another example, we could have better incorporated aspects of authenticity (promoting Situated Learning) by including more random events (a characterizing feature of the real world) in our models. These types of events are only used sparingly in many of our models.

Like most software engineering educational approaches, SimSE was not designed with learning theories in mind. However, by looking back on its design in light of learning theories, we have learned a great deal about how SimSE promotes learning and how it can be improved to foster greater learning, as we have seen in this section.

Learning Theory-Centric Evaluation

Although we did not explicitly use learning theories in SimSE's initial design, we did use them as a central guiding factor in designing a major part of its evaluation. Validating that the theories an approach was designed to employ (or appear to employ) are *actually* employed, as well as discovering if an approach incorporates aspects of any additional theories, can be highly useful exercises—such data can be used to make that and other similar approaches more effective as they are tailored to exploit the characteristics known to promote each theory (van Eck, 2006). Thus, as part of SimSE's evaluation, we performed an in-depth observational study that focused on investigating the learning processes of SimSE players to determine whether they exhibited behaviors indicative of various learning theories.

Case Study: SimSE Evaluation Setup

For this study, we used as subjects 11 undergraduate students who had passed the introductory software engineering course at the University of California, Irvine. This requirement was put in place so that they would have at least the basic understanding of software engineering concepts required to play SimSE. The study occurred in a one-on-one setting—one subject and one observer. Each subject was first given instruction on how to play SimSE, and was then observed playing SimSE for about 2.5 hours. In order to evaluate how well the explanatory tool achieves its goal of aiding Learning through Reflection, we had eight students play SimSE with the explanatory tool and three without. (Differences in the behavior, attitudes, and opinions of each group could then be compared, though clearly, not to the extent of being statistically significant.) While subjects were playing, their game play and behavior were observed and noted. Following this, the subject was interviewed about their experience for about 30 minutes. In addition to any spontaneous questions the observer formulated based on a particular subject's actions or behavior during game play, all subjects were asked a set of standard questions. Several of these questions were designed to specifically detect the presence of one or more learning theories in the subject's learning process. Some questions did not target a particular theory or set of theories, but were instead meant to evoke insightful comments from the subject from which various learning theories could be inferred, and from which general insights into the learning process could be discovered. Some samples from the standard set of questions are listed here, with the targeted learning theory (or theories) listed in parentheses afterwards when applicable.

- *To what do you attribute the change (or lack of) (improvement, worsening, fluctuation, steady state) of your score with each game?* (Discovery Learning, Learning through Failure)
- *Do you feel you learned more when you "won" or when you "lost"? Why? What did you learn from each "win" or "loss"?* (Discovery Learning, Learning through Failure)

- *When you lost, did you feel motivated to try again or not? Why?* (Learning through Failure)
- *On a scale of 1 to 5, how much did playing SimSE engage your attention? Why?* (Keller's ARCS)
- *How much has your level of confidence changed in the learning material since completing this exercise?* (Keller's ARCS)
- *Did you feel that you learned any new software process concepts from playing SimSE that you did not know before? If so, which ones?* (answer could be indicative of multiple theories)
- *If you feel you learned from SimSE, what do you believe it is about SimSE that facilitated your learning?* (answer could be indicative of multiple theories)

There were also some questions primarily designed for comparison between the subjects who used the explanatory tool and those who did not. These questions were aimed at discovering how the player went about figuring out the reasoning behind their scores, as well as how well they understood this reasoning.

- *Where do you think you went wrong in game 1/2/x?* (Learning through Reflection)
- *Please describe the process that you followed to figure out the reasoning behind your score, or where you went wrong/right.* (Learning through Reflection)

Following the experiment, the interviewer's observations and interview notes were analyzed to try to discover which behaviors and comments were indicative of the various learning theories, and how, as well as to discover any other insights about SimSE as a teaching tool that could be gained from this data.

Evaluation Results

The learning theory that was most clearly involved in every subject's learning process was Discovery Learning. All subjects were able to recount at least a few lessons they learned from SimSE, and none of these lessons were ever told to them explicitly during their experience. Rather, they discovered them independently through exploration and experimentation within the game. Interestingly, although all subjects that played a model seemed to discover the same lessons (for the most part), no two subjects discovered them in the same way. Every subject approached the game with a different strategy, but came away with similar new knowledge, suggesting that SimSE can be applicable to a wide range of students that come from different backgrounds with different ideas and possibly, different learning styles. This is a central aspect of a student-centered theory like Discovery Learning. Since learning depends primarily on the learner and not the instructor, the learner is free to use their own style and ideas in discovering the knowledge, rather than being forced to adhere to a rigid style of instruction.

Learning through Failure also seemed to be widely evident. Every subject seemed to take a "divide and conquer" approach to playing SimSE, isolating aspects of the model and tackling them individually (or a few at a time). When subjects described the progression of their games in the interviews, it was clear that the way they conquered each aspect was by going through at least one or two rounds of failure in which they discovered what *not* to do, and from this discovering a correct approach that lead to success. When asked explicitly about learning through failure, every subject stated that they learned when they failed, but the amount of learning they reported varied. Five subjects said they learned more from failure than success, two subjects said they learned more when they succeeded, and four subjects said they learned equally as much from failure and success. All but one subject said that they were motivated to try again after they failed. This motivation was also evident in the behavior of several subjects, as some, after the completion of one failed game, hurriedly and eagerly started a new one. One subject

even tried to start a new game when the time for the game play portion of the experiment was up and he was already informed that it would be the last game.

The Learning by Doing theory seemed to be involved in most of the subjects' learning experience. Eight out of the 11 subjects made comments about their experience playing SimSE that hinted at aspects of Learning by Doing. Some of their comments included:

- "[SimSE helped me learn because it] *puts you in charge of things. It's a good way of applying your knowledge.*"
- "[SimSE helped me learn because it is] *interactive, not just sitting down and listening to something.*"

Comments indicative of Situated Learning were also rather frequent, mentioned by seven out of the 11 subjects. Some of these included:

- "[SimSE helped me learn because] *it was very realistic and helped me learn a lot of realistic elements of software engineering, such as employees, budget, time, and surprising events.*"
- "[One of the learning-facilitating characteristics of SimSE was] *seeing a real-life project in action with realistic factors like employee backgrounds and dialogues.*"

Behaviors and comments suggestive of Keller's ARCS Motivation Theory were also evident, although certain aspects of the theory came out stronger than others. To explain, let us look at the four aspects of the theory (attention, relevance, confidence, and satisfaction) individually.

First, the attention of the subjects seemed to be quite engaged with SimSE. This was evident in their body language, the comments made both during game play and the interview, and their ratings of SimSE's level of engagement. Many of them spent the majority of their time during game play sitting on the edge of their seats, leaning forward and fixing their eyes on the screen. There were head nods, chuckles in response to random events and character descriptions, shouts of "Woo hoo!" after achieving a high score in a game, shaking of the head when things were not going so well for a player, and requests of, "Can I try this one more time?" when the experiment's allotted time for game play was coming to an end. Words some subjects used to describe SimSE in the interview were "challenging", "fun", "interesting", "addictive", and "amusing." When explicitly asked how much SimSE engaged their attention, the students rated it quite high—4.1 on average out of five.

Second, relevance was rated moderately high, but not as high as level of engagement. Five of the subjects rated SimSE's relevance to their future experiences as "pretty relevant" or "very relevant", five described it as "somewhat" or "partially" relevant and one said it was not relevant at all. Although not explicitly asked about SimSE's relevance to their past experiences, nearly all of the subjects mentioned that they used some of the knowledge they had learned in software engineering courses to come up with their strategies for playing the game, suggesting that there is also a relevance between their past experiences (learning the concepts in class) and their learning experience with SimSE.

Third, most subjects felt their level of confidence in the learning material (the software process model simulated and software process in general) had increased at least somewhat since playing SimSE. Four subjects reported their level of confidence had changed "a lot" or "very much", five said it had changed "somewhat", and two said it had not changed at all.

Fourth, satisfaction was rated quite high by the subjects. Nine out of the 11 subjects reported that they were "quite satisfied", "very satisfied", "fully satisfied", or "pretty satisfied", and three subjects stated they were "somewhat satisfied." Most of the reported factors that contributed to a feeling of satisfaction pertained to a subject's increasing success from game to game, although

some also mentioned that the sheer fun and challenge of SimSE contributed to their satisfaction as well.

The explanatory tool did seem to promote Learning through Reflection, to some extent. Most of the subjects that had access to the explanatory tool did make use of it, the duration of its use after most games ranging from five to 25 minutes. It was obvious that the subjects who did not have the explanatory tool (to whom we will henceforth refer as "non-explanatory subjects") were significantly more confused and less confident about the reasoning behind their scores and how to improve than those who did have the explanatory tool (to whom we will henceforth refer as "explanatory subjects"). All of the non-explanatory subjects expressed this, while only one explanatory subject stated such an opinion. The following are some of the comments made by the non-explanatory subjects:

- "*I was trying to guess what I was doing wrong, so I probably chose the wrong areas that I was doing wrong, and then I tried to switch back to my original way and then I kind of forgot what that was and once I started trying to improve it, all of my little details started changing and I didn't know what parts were causing my score to go lower*."
- "*I felt like I knew, oh, that's where I went wrong sometimes, like I should spend a little less time there, but a lot of times I was wrong about where it was I went wrong*."

On the other hand, most of the explanatory subjects' comments expressed that the explanatory tool did, indeed facilitate their learning:

- "[The explanatory tool] *showed me why I was doing poorly—because of certain events that were happening."*
- "*The rules* [described in the explanatory tool] *are really helpful—even if someone doesn't know anything about software engineering I think the rules can teach you how to play the game*."

Implications of Evaluation Results

Evaluating SimSE in terms of learning theories provided us with several valuable insights into how SimSE helps students learn. In addition, it also helped us to discover ways to potentially make SimSE more effective. In this subsection, we describe how focusing on some of the theories in our evaluation provided us with knowledge that will help us maximize SimSE's effectiveness.

Learning through Failure: Overall, the challenge of receiving a "failing" score and trying to improve it seemed to be a significant avenue of learning and a strong motivating factor of SimSE. This reinforced our notion that simulation models should be made challenging enough that students are set up to fail at times. It is these failures that provide some of the greatest opportunities for learning. By focusing on this aspect in our observations, we also discovered that one of our models (Rapid Prototyping) was not quite challenging enough, and students could sometimes get a good score without really learning the lessons. Thus, we have since added more challenges to this model, and will continue to build simulation models in the future that have an adequate level of challenge.

Learning by Doing: Several of the subjects' comments mentioned the ability to put previously learned knowledge into practice as a learning-facilitating characteristic of SimSE. This validates our choice to use SimSE complementary to other teaching methods, so that it can fulfill this important role of being an avenue through which students can employ Learning by Doing as they do the things they only heard about in class.

Situated Learning: The realistic elements in SimSE seem to add significantly to its educational effectiveness. Thus, it is important that we continue to include elements of the real world in our models, in order to situate students' knowledge in a realistic environment.

Elaboration: It became clear from our observations that one of our models (waterfall) is much too large and complex for a "SimSE beginner." (Although the waterfall process is a simple one, the corresponding SimSE model is quite complicated, incorporating several non-technical, managerial aspects.) By giving such a complex model to a student who has never played SimSE before, we were clearly violating the principles of the Elaboration theory. Thus, viewing this result in light of that theory taught us that such a model should not be introduced until the student has played other, simpler models first.

Keller's ARCS: Through this study we were able to discover what elements of SimSE and its models best hold students' attention by noting when students appeared to be most engaged, and what kinds of things they commented about favorably in the interviews. For example, several students mentioned that the random events in the models (e.g., the customer changing their mind and requiring the team to rework part of the code) added an element of surprise and realism that kept things entertaining. Thus, we will continue to build these elements into our future models, as well as try to maximize them in our current models. We also discovered which elements students found un-engaging. For instance, several subjects thought the inspection model was boring and repetitive. Through the interviews, we were able to detect exactly what it was about the inspection model that made it this way, and have recently implemented changes that we anticipate will make it more interesting for future SimSE players.

Learning through Reflection: The explanatory tool partially fulfills its goal of facilitating reflection, but it is clear that it needs to be improved. In particular, more help needs to be given to the user in generating meaningful, useful graphs, and the rule descriptions need to be more easily accessible. We have recently addressed these issues in our development by adding attributes to each model that are meant specifically for explanatory graphing purposes and by making the rule descriptions more accessible through the user interface.

Learning theories can help structure evaluations by providing ideas about what the researcher should be looking for in the learning processes of students. As we have seen with SimSE, this can be done even if the approach was not designed with learning theories in mind. A careful retro-analysis of the approach's design in terms of learning theories can reveal the aspects that a learning theory-centric evaluation should focus on. Conducting such an evaluation has the potential to both reveal the effectiveness of an approach, as well as guide future work in the area.

Certainly, not every aspect of an approach can be evaluated this way—an evaluation focused on learning theories should only be one part of an evaluation plan. In addition to the evaluation described here, SimSE's evaluation plan also included a pilot study, a comparative study, and in-class studies, each of which was designed to evaluate different aspects of SimSE to form a comprehensive picture of its ability as a teaching tool (see (Navarro & van der Hoek, 2007) for more information about these studies).

Summary

Learning theories are an important educational resource of which the software engineering educational community has not yet taken full advantage. Learning theories can be used to categorize, design, evaluate, and communicate about software engineering educational approaches, providing a structured and informed way to move our domain forward with approaches that are effective and well-understood. We have shown one example of applying

learning theories to software engineering education in our analysis and evaluation of SimSE. It is our hope that educators can take this example and apply it to other approaches and areas of software engineering education to create more effective teaching strategies that are rooted in educational theory.

More Information

More information about SimSE, including downloads, evaluations, and publications, are available at http://www.ics.uci.edu/~emilyo/SimSE/.

Acknowledgements

We would like to thank the reviewers of this chapter for their highly useful and constructive feedback. Effort partially funded by the National Science Foundation under grant number DUE-0618869.

References

Abernethy, K., & Kelly, J. (2000). Technology transfer issues for formal methods of software specification. In S. A. Mengel & P. J. Knoke (Eds.), *Proceedings of the thirteenth conference on software engineering education and training* (pp. 23-31). Austin, TX: IEEE Computer Society.

Baker, A., Navarro, E. O., & van der Hoek, A. (2003). Problems and programmers: An educational software engineering card game. In *Proceedings of the 2003 international conference on software engineering* (pp. 614-619). Portland, Oregon.

Beckman, K., Khajenoori, K., Coulter, N., & Mead, N. R. (1997). Collaborations: Closing the industry-academia gap. *IEEE Software, 14*(6), 49-57.

Blake, B. M. (2003). A student-enacted simulation approach to software engineering education. *IEEE Transactions on Education, 46*(1), 124-132.

Bruner, J. S. (1967). *On knowing: Essays for the left hand*. Cambridge, Mass.: Harvard University Press.

Collofello, J. S. (2000). University/industry collaboration in developing a simulation based software project management training course. In S. Mengel & P. J. Knoke (Eds.), *Proceedings of the thirteenth conference on software engineering education and training* (pp. 161-168). Austin, TX: IEEE Computer Society.

Dantas, A. R., Barros, M. O., & Werner, C. M. L. (2004). A simulation-based game for project management experiential learning. In *Proceedings of the 2004 international conference on software engineering and knowledge engineering*. Banff, Alberta, Canada.

Dawson, R. (2000). Twenty dirty tricks to train software engineers. In *Proceedings of the 22nd international conference on software engineering* (pp. 209-218): ACM.

Dewey, J. (1916). *Democracy and education*. New York, NY: Macmillan.

Drappa, A., & Ludewig, J. (2000). Simulation in software engineering training. In *Proceedings of the 22nd international conference on software engineering* (pp. 199-208): ACM.

Favela, J., & Pena-Mora, F. (2001). An experience in collaborative software engineering education. *IEEE Software, 18*(2), 47-53.

Ferrari, M., Taylor, R., & VanLehn, K. (1999). Adapting work simulations for schools. *The Journal of Educational Computing Research, 21*(1), 25-53.

Fishbein, M., & Ajzen, I. (1975). *Belief, attitude, intention, and behavior: An introduction to theory and research*. Reading, Mass.: Addison-Wesley.

Gibbons, A. S. (2001). Model-centered instruction. *Journal of Structural Learning and Intelligent Systems, 14*(4), 511-540.

Gnatz, M., Kof, L., Prilmeier, F., & Seifert, T. (2003). A practical approach of teaching software engineering. In P. J. Knoke, A. Moreno & M. Ryan (Eds.), *Proceedings of the sixteenth conference on software engineering education and training* (pp. 120-128). Madrid, Spain: IEEE.

Goold, A., & Horan, P. (2002). Foundation software engineering practices for capstone projects and beyond. In M. McCracken, M. Lutz & T. C. Lethbridge (Eds.), *Proceedings of the fifteenth conference on software engineering education and training* (pp. 140-146). Covington, KY, USA: IEEE.

Groth, D. P., & Robertson, E. L. (2001). It's all about process: Project-oriented teaching of software engineering. In D. Ramsey, P. Bourque & R. Dupuis (Eds.), *Proceedings of the fourteenth conference on software engineering education and training* (pp. 7-17). Charlotte, NC, USA: IEEE.

Halling, M., Zuser, W., Kohle, M., & Biffl, S. (2002). Teaching the unified process to undergraduate students. In M. McCracken, M. Lutz & T. C. Lethbridge (Eds.), *Proceedings of the fifteenth conference on software engineering education and training* (pp. 148-159). Covington, KY, USA: IEEE.

Hayes, J. H. (2002). Energizing software engineering education through real-world projects as experimental studies. In M. McCracken, M. Lutz & T. C. Lethbridge (Eds.), *Proceedings of the fifteenth conference on software engineering education and training* (pp. 192-206). Covington, KY: IEEE.

Hilburn, T. (1999). PSP metrics in support of software engineering education. In H. Saiedian (Ed.), *Proceedings of the twelfth conference on software engineering education and training* (pp. 135-136). New Orleans, LA, USA: IEEE.

Keller, J. M. (1983). Motivational design of instruction. In C. M. Reigeluth (Ed.), *Instructional design theories and models: An overview of their current status*. Hillsdale, NJ: Erlbaum.

Kirschner, P. A., Sweller, J., & Clark, R. E. (2006). Why minimal guidance during instruction does not work: An analysis of the failure of constructivist, discovery, problem-based, experiential, and inquiry-based teaching. *Educational Psychologist, 41*(2), 75-86.

Knowles, M. (1984). *Andragogy in action: Applying modern principles of adult education*. San Francisco, CA: Jossey Bass.

Kolb, D. A. (1984). *Experiential learning: Experiences as the source of learning and development*. Englewood Cliffs, NJ, USA: Prentice-Hall International, Inc.

Kolikant, Y. B. (2001). Gardeners and cinema tickets: High school students' preconceptions of concurrency. *Computer Science Education, 11*(3), 221-245.

Kornecki, A. J. (2000). Real-time computing in software engineering education. In S. A. Mengel & P. J. Knoke (Eds.), *Proceedings of the thirteenth conference on software engineering education and training* (pp. 197-198). Austin, TX, USA: IEEE.

Kornecki, A. J., Khajenoori, S., & Gluch, D. (2003). On a partnership between software industry and academia. In P. J. Knoke, A. Moreno & M. Ryan (Eds.), *Proceedings of the sixteenth conference on software engineering education and training* (pp. 60-69). Madrid, Spain: IEEE.

Lave, J. (1988). *Cognition in practice: Mind, mathematics, and culture in everyday life*. Cambridge, UK: Cambridge University Press.

McKim, J. C., & Ellis, H. J. C. (2004). Using a multiple term project to teach object-oriented programming and design. In T. B. Horton & A. E. K. Sobel (Eds.), *Proceedings of the seventeenth conference on software engineering education and training* (pp. 59-64). Norfolk, VA: IEEE.

Navarro, E. O. (2005). *A survey of software engineering educational delivery methods and associated learning theories* (Technical Report No. UCI-ISR-05-5). Irvine, CA: University of California, Irvine.

Navarro, E. O. (2006). *SimSE: A software engineering simulation environment for software process education.* Ph.D. Dissertation, University of California, Irvine, Irvine, CA.

Navarro, E. O., & van der Hoek, A. (2005a). Design and evaluation of an educational software process simulation environment and associated model. In T. C. Lethbridge & D. Port (Eds.), *Proceedings of the eighteenth conference on software engineering education and training*. Ottawa, Canada: IEEE.

Navarro, E. O., & van der Hoek, A. (2005b). Scaling up: How thirty-two students collaborated and succeeded in developing a prototype software design environment. In T. C. Lethbridge & D. Port (Eds.), *Proceedings of the eighteenth conference on software engineering education and training*. Ottawa, Canada: IEEE.

Navarro, E. O., & van der Hoek, A. (2007). Comprehensive evaluation of an educational software engineering simulation environment. In H. Edwards & R. Narayanan (Eds.), *Proceedings of the twentieth conference on software engineering education and training*. Dublin, Ireland.

Nulden, U., & Scheepers, H. (2000). Understanding and learning about escalation: Simulation in action. In *Proceedings of the 3rd process simulation modeling workshop (prosim 2000)*. London, United Kingdom.

Ohlsson, L., & Johansson, C. (1995). A practice driven approach to software engineering education. *IEEE Transactions on Education, 38*(3), 291-295.

Pfahl, D., Klemm, M., & Ruhe, G. (2000). 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.

Reigeluth, C. M., & Rodgers, C. A. (1980). The elaboration theory of instruction: Prescriptions for task analysis and design. *NSPI Journal, 19*, 16-26.

Roblyer, M. D. (2005). *Integrating educational technology into teaching* (4th ed.). Upper Saddle River, NJ: Prentice Hall.

Schank, R. C. (1997). *Virtual learning*. New York, NY, USA: McGraw-Hill.

Schön, D. (1987). *Educating the reflective practitioner*. San Francisco, CA, USA: Jossey-Bass.

Sharp, H., & Hall, P. (2000). An interactive multimedia software house simulation for postgraduate software engineers. In *Proceedings of the 22nd international conference on software engineering* (pp. 688-691): ACM.

Stevens, S. M. (1989). Intelligent interactive video simulation of a code inspection. *Communications of the ACM, 32*(7), 832-843.

Tomayko, J. E. (1996). Carnegie Mellon's software development studio: A five year retrospective. In *Proceedings of the ninth conference on software engineering education and training* (pp. 119-129). Daytona Beach, FL, USA: IEEE.

van Eck, R. (2006). Digital game-based learning: It's not just the digital natives who are restless. *Educause Review, 41*(2), 17-30.
Wohlin, C., & Regnell, B. (1999). Achieving industrial relevance in software engineering education. In H. Saiedian (Ed.), *Proceedings of the twelfth conference on software engineering education and training* (pp. 16-25): IEEE Computer Society.