Collaborative Learning as Interplay between Simulation Model Builder and Player

Kurt Schneider[1] and Kumiyo Nakakoji[1,2]

[1]*Department of Computer Science and Institute of Cognitive Science, University of Colorado at Boulder, USA*
[2]*Software Engineering Laboratory, Software Research Associates, Inc., Japan*

Abstract

We present a simulation-based learning environment that supports collaborative learning as interplay between a model builder and a player. The SESAM system serves as a medium through which a model-builder and a player communicate understanding and experience about an evolving model.

Keywords — simulation-based learning, model-builder player collaboration, collaborative environment.

1. Introduction

A simulation-based learning environment SESAM (Software Engineering by Simulation of Animated Models) has been developed to investigate software engineering concepts and how to apply these concepts to practical situations. SESAM simulates the development cycle of a software development project, allowing each player to participate in the role of project manager. The system simulates software artifacts, various types of documents, fictitious participants (including client and project members), and the project evolution over time based on a model constructed by a model-builder.

Our approach integrates simulation model-building and playing with a learning environment rather than separating the two activities. Mediated by SESAM, a model builder and a player collaboratively analyze software engineering phenomena by constructing, validating, and refining a model of a software project through a cycle of four processes: (1) a model builder constructs a model; (2) a player interacts with a simulation based on the model; when the player experiences difficulties, he or she annotates simulation runs with textual remarks and questions; (3) the model builder analyzes the recorded simulation run; and (4) the model builder and the player examine the run together. The system's graphical interface allows model builders to construct models via direct-manipulation interaction. A text-based menu-driven interface allows players to interact with the simulation (see Figure 2).

Although we present our work in the domain of software engineering education, we expect the approach and its computational substrate to be applicable to many educational domains in which models are still evolving.

2. The Model Builder and Player Approach for Simulation-Based Learning Environments

Models are formal representations that highlight specific aspects of a much larger and more complex problem. Models omit details not relevant to the current task and help people focus on problems at hand in "practical" situations.

Most simulation-based educational systems [4, 9] do not support the evolution and refinement of underlying simulation models. Many systems are based on the assumption that an appropriate model already exists in the domain. Consequently, such systems completely separate model building activities from simulation-playing activities.

In rapidly changing work domains, however, the underlying model easily becomes obsolete. In domains such as software engineering, for instance, building a simulation model itself is a research activity. There is little qualitative information available that is precise and unambiguous enough to base a model on. Models, rules – any explicit representations can never completely cover reality [10]; some parts of current affairs always remain tacit [6]. In these domains, not only students, but the model-builders themselves need to learn about the domain by refining the model.

To cope with this problem, a simulation-based learning environment needs to support collaborative learning as the interplay between a model builder and a player. This changes the view of collaborative learning from the teacher-student relationship to the model builder-player relationship. As illustrated in Figure 1, a model builder and a player interact through a simulation-based collaborative learning environment.

The simulation presents the consequences of the player's interaction based on an underlying model. The player may make questionable decisions, or make mistakes that are anticipated by the model, which lead to consequential problems within the game. Making mistakes is a great opportunity for a player to learn. Explanations and information relevant to the problem can be provided, being contextualized within the simulated situation that the player is directly engaged in [2]. At the same time, it is possible that a player's actions reveal a deficiency in the underlying model, which leads to implausible model behavior. Such breakdown situations provide a model builder with opportunities to analyze and refine the model by exposing tacit aspects of practice. Thus, the model builder also learns.

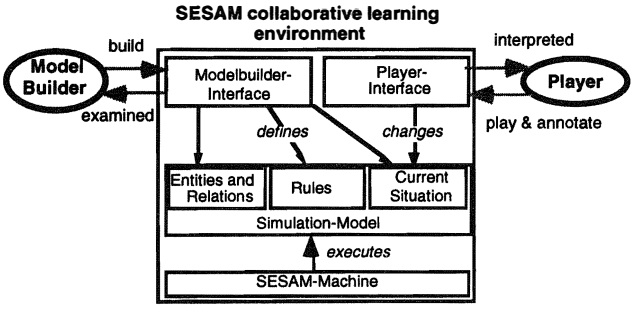

Figure 1. Collaborative Learning between a Model Builder and a Player through SESAM.

Figure 1 illustrates the SESAM architecture. The *SESAM Machine* interprets graphical models, produces textual simulation runs, automatically proceeds model time, applies effect models whenever appropriate, keeps track of model evolution and triggers pseudo-random events.

3. SESAM

SESAM was developed at the University of Stuttgart (Germany) as a tool for software engineering research and education. The system is written in Smalltalk 4.1 and runs on several different platforms.

A SESAM player manages a project by planning the project, updating those plans, and then enacting the plans in fine-grained daily decisions. As project manager, players act on and react to changing situations represented by fictitious artifacts and stakeholders (clients and project members). Project evolution is modeled based on the software quantum metaphor [1, 7], which models functional completeness of software artifacts and documents quantitatively. Pre-stored statements from fictitious team members are triggered and displayed according to the project state and player's interaction. The goal of the player is "*to produce required functionality and quality within the budget and schedule by motivating team members, by assigning them to adequate tasks, and by involving the client to validate the evolving requirements.*" In what follows, we present a scenario to describe how a player interacts with the system through the player interface, followed by a brief overview of the model builder interface. See [3, 7, 8] for details.

3.1. Scenario

When Jane, a computer science student, first invokes SESAM, a short project description is presented in the *What Happened* window (see Figure 2-(a)). Although the given deadline and budget partially indicate how to plan her project, she lacks important details to estimate her project's effort. She has to take some actions using the *Simulator* window.

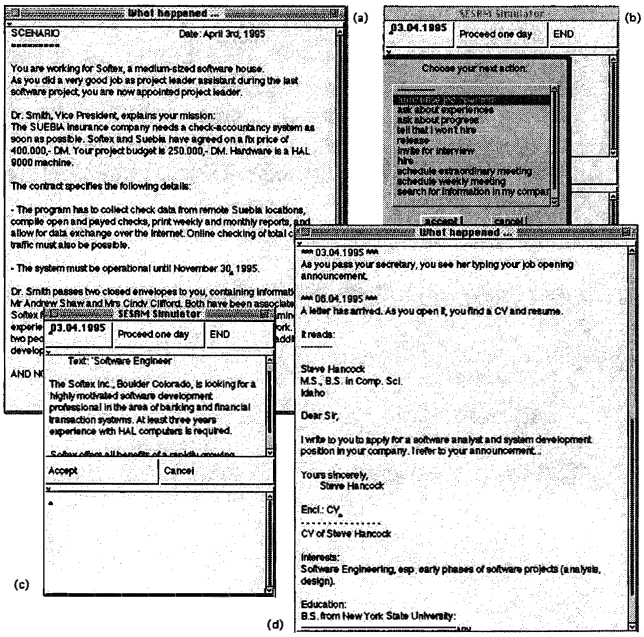

Figure 2. Screen Images of SESAM.

Jane finds a long menu of possible actions (see Figure 2-(b)). Some of them (e.g. "finish project") are obviously inadequate. Jane first wants to hire team members. She selects the "*announce job opening*" entry, and the system asks her to specify the exact text of her job opening (see Figure 2-(c)).

After she completes the job-opening specification, SESAM executes the simulation according to her action. At first nothing happens. Jane notices that the model date (see the upper left corner of the *Simulator* window in Figure 2-(c)) has been incremented by one working day. Writing the job opening costs her one day. Jane takes several more actions; she introduces herself to the customer, plans and looks for information in the company. Some simulated days later (some minutes later in real time), a resume and CV appears on the screen. *Steve Hancock* applies for the job (see Figure 2-(d)). Jane invites Steve for a job interview and

decides to hire him. Like all other actions, job interviews and hiring a developer take model time.

Later in the project, Jane finds herself left alone with very little feedback from Steve and the other team members she hired. In meetings, she gets only superficial pieces of information on project progress (e.g., "We *have about 10,000 lines of code*"). Jane gets frustrated and complains about this in an annotation: "*How can I estimate cost and effort, if I have no precise status information?" "What is the quality of my product? "* and *"What does it mean to have 10,000 loc?"* Desperate, she invites the customer for a prototype review. To her surprise, the customer is not satisfied and requests several modifications. Now, all of Jane's plans must be updated, all documents (specification, design, code) must be reworked. Soon it becomes clear that the customer is good at finding flaws. At some point, however, Jane must decide to nevertheless continue development; otherwise she will risk schedule and budget slippage.

After Jane finishes her simulation game, Jane's advisor, Bill, reviews Jane's simulation. Because of her annotation, Bill finds that Jane was frustrated by the sparing feedback from her team members. Bill also finds that Jane's idea of inviting the customer was a good action, as it allowed Jane to assess product quality. Bill explains to Jane that in reality, no one tells project managers about their product quality, either. Jane and Bill talk about quality assessment techniques that Jane could have used. Jane claims a real software developer would have given at least some hint of the degree of work completion; the fictitious team member Steve did not. Bill agrees with her comment and considers model extensions.

3.2. Model Builder Interface

The system uses three types of representations for a simulation-model: entity-relationship model, rules, and particular situations. The model builder interface provides three semi-graphical editors accordingly. Building models does not require any significant programming.

The *Entity-Relationship (ER) model* schema (see Figure 3-(a)) defines entities and relationships that constitute the model world. In the *situation model,* the initial setup for the simulation is created by instantiating the entity types (see Figure 3-(b)). A textual description of this situation represents the context to the player. *Effect model (rules)* are used to describe dynamic behavior of simulated project participants, evolution of documents and artifacts (not shown).

4. Discussions

SESAM allows both model builders and players to tie their experiences and arguments to a common medium: the model and its simulated events. To foster this collaboration both in indirect (asynchronous) and face-to-face (synchronous) modes, the system keeps a record of player actions together with model time stamps to trace what has happened during the simulation. The accumulated data (player actions, invisible model behavior, and final project results) is a basis for discussing a simulation run in detail. When a run is reproduced based on the interaction protocol, additional information can be captured and displayed, including attribute value plots, effect model activation and deactivation information. Model builder and player may also make modifications to the model and rerun it on the fly to see the consequences.

SESAM mediates collaboration by making models accessible from two different perspectives: (1) a model builder initially accesses phenomena through the semi-graphical model notation, and (2) a player refers to textually represented events that occurred during the game. The model notations used in SESAM are a graphical, though *formal* representation of a project

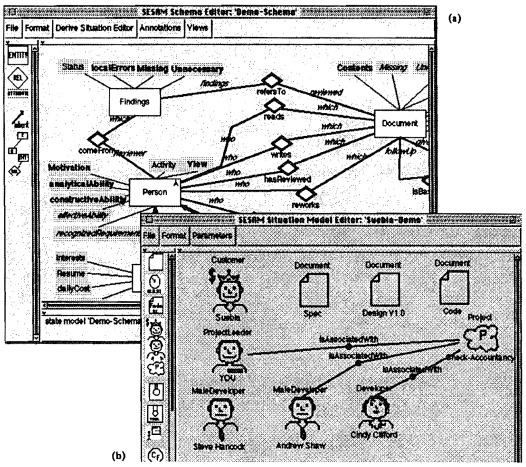

Figure 3. ER schema (a) and a situation graph (b) of a model used in the scenario.

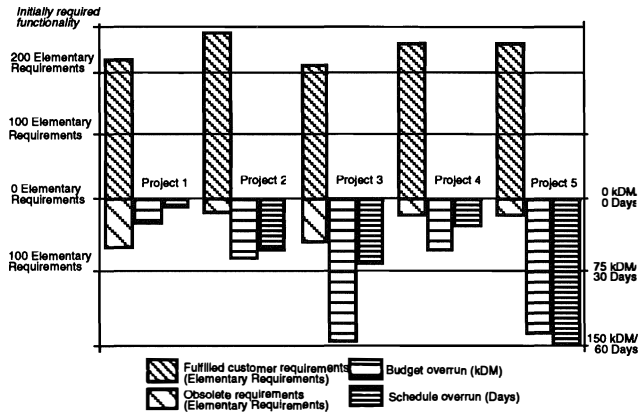

Figure 4. Overall results of the project management course.

model. *Informal* annotation is provided to complement those formal model aspects to provide expressiveness to players. Mediated by SESAM, model builders and players discuss questions such as "*does this effect really occur in software projects?*" "*what are the reasons, and how are they represented in the model?*" and "*why is it important to know this effect and to integrate it in an educational model?*"

A study of SESAM with a graduate level course at the University of Stuttgart has demonstrated that the approach was effective both for students (served as players) and for faculties (served as model builders). In the study, the model used in the above scenario was presented as an adventure game to 10 graduate students of computer science in a project management course. All students were asked to play the same simulated project in independent, parallel games. As illustrated in Figure 4, results differed dramatically in terms of quality and resource consumption due to different management. Project distortions were tracked back to early planning or scheduling mistakes. The project which performed best (Project 2) had a project plan which was constantly updated. Project 3 performed worst; it spent a lot of effort unsystematically, which resulted in large overrun on budget and the worst quality delivered. For detailed description of the model and the course, see [1].

The upper part of Figure 4 represents functional completeness, which should be as high as possible. The lower part gives the amount of obsolete functionality, the budget overrun in 1000 DM, and the schedule overrun in days, the values of which should be as low as possible.

5. Conclusion

We view learning as collaboration between a model builder and a player mediated by computational environments. Software engineering knowledge is scattered. There is no concise, comprehensive model that is accepted by a wide range of researchers and practitioners. Software engineering textbooks convey isolated, mostly vague pieces of advice, due to the general scope and audience they are written for.

Traditional instructionist knowledge transfer approach is inadequate for such educational domains [5]. SESAM provides two perspectives on a single evolving model. The model builder perspective offers semi-graphical interfaces for constructing a model to be used for simulation. Players experience the model's dynamic behavior in an adventure-game style through the player perspective. The system records the simulation runs, and players can annotate the simulation with critiques and questions. Our assessment studies have demonstrated that relatively simple models provided opportunities to learn for both model builders and players – in both reflective and experiential modes.

Acknowledgments

We would like to thank the software engineering research group at the University of Stuttgart and its head, Jochen Ludewig, who initiated the SESAM project. We also thank Gerhard Fischer and the members of the lifelong learning and design (L3D) center at the University of Colorado. We thank Brent N. Reeves for constructive comments. This research was supported by the German Academic Exchange Service (DAAD) and Software Research Associates, Inc., Japan.

References

1. Deininger, M.; Schneider, K. (1994): Teaching Software Project Management by Simulation - Experiences with a Comprehensive Model; Proceedings. of the Conference on Software Engineering Education (CSEE), Austin, Texas, Jan. 1994

2. Fischer, G. (1994): Conceptual Frameworks and Computational Environments in Support of Learning on Demand. in: C. Hoyles (ed.): The Design of Computational Media to Support

Explanatory Learning. Springer-Verlag, New York, NY.

3. Ludewig, J. (1993): Problems in modeling the software development process as an adventure game. in H.D. Rombach, V.R. Basili, R.W. Selby (eds.):Experimental Software Engineering Issues: Critical Assessment and Future Directions. Springer-Verlag, Berlin usw., Lecture Notes in Computer Science 706, pp 23-26.

4. McKeeman, W.M. (1989): Graduation Talk at Wang Institute, IEEE Computer, vol. 22, no. 5

5 Papert, S., Harel, I. (eds, 1993): Constructionism. Ablex Publishing Corp., Norwood, NJ

6. Polanyi, M. (1966): The Tacit Dimension. Doubleday, Garden City, NY

7. Schneider, K. (1994): Executable Models of Software Development (in German: Ausführbare Modelle der Software-Entwicklung: Struktur und Realisierung eines Simulationssystems); Dissertation University of Stuttgart; vdf Hochschulverlag, Zürich

8. Schneider, K. (1995): Dynamic Pattern Knowledge in Software Engineering. Proceedings of the SEKE'95 Conference to be held in Rockville, MD; June 22-24, 1995

9. Tambe, M., Johnson, W.L., Jones, R.M., Koss, F., Laird, J.E., Rosenbloom, P.S., Schwamb, K. (1995): Intelligent Agents for Interactive Simulation Environments, AI Magazine, American Association for Artificial Intelligence, Vol.16, No.1, pp.15-40.

10. Winograd, T., Flores, F. (1986): Understanding Computers and Cognition: A New Foundation for Design. Ablex Publishing Corporation, Norwood, NJ

Authors' Addresses

Kurt Schneider and *Kumiyo Nakakoji*: Center for Life-Long Learning and Design, Department of Computer Science and Institute of Cognitive Science, University of Colorado, ECOT 7-7, Campus Box 430, Boulder CO 80309-0430, USA