Model Driven Game Development: Experience and Model Enhancements in Software Project Management Education

Alexandre R. Dantas, Gustavo O. Veronese, Márcio de O. Barros, Cláudia M.L. Werner

1 COPPE - Computer Science Department
   Federal University of Rio de Janeiro
   Caixa Postal: 68511 - CEP: 21945-970
   Rio de Janeiro - Brazil
   Voice: +55 21 2562-8675
   Fax: +55 21 2562-8676
   {alexrd, veronese, werner}@cos.ufrj.br

2 Applied Informatics Department
   Federal University of Rio de Janeiro State
   Av. Pasteur 458, Urca
   CEP: 22290-240
   Rio de Janeiro – Brazil
   Voice: +55 21 2244-5613
   marcio.barros@uniriotec.br

Abstract

The inadequate use of management techniques in software projects is usually associated to the lack of efficient strategies to teach managers and prepare them to face the challenges imposed by real world projects. Many studies have proposed that a learning-by-doing educational approach would be more effective than the traditional lecture-oriented, professor-centered strategy. However, the former approach requires an environment where students can act as managers without the cost and risks associated to project failures. Games are proposed to act as such an environment, emulating reality through graphical or textual constructs and allowing the trainees to play the managers’ role in a risk-free environment.

In this work, we present our experiences in developing System Dynamics based games for software project management. We describe a project management game intended for training purposes and the changes that were made to allow a System Dynamics simulator to support game-like interactions. Moreover, we describe an experimental evaluation of the game’s application as a learning-by-doing environment for management students. Based on the experience we acquired by building such interface, we propose models to describe the story underlying a game and its graphical presentation. Such models allow the construction of games without programming, thus fastening the development of simulation-based games.

KEYWORDS: Game-based Education, System Dynamics, Project Management

1. Introduction

Many studies have observed that the adoption of project management techniques in software development projects is usually deficient and inadequate (The Standish Group, 2003, Mandl-Striegnitz, Lichter, 1998). Such studies suggest that these deficiencies may be a potential cause for so many projects experiencing poor quality, schedule and budget overruns. On the other hand, the inadequate use of project management practices and techniques can be related to the lack of comprehension of how these elements can be applied in real situations.
Project management is a knowledge-intensive activity. Personal expertise is a major success factor for managers. However, the common practice of promoting new managers from the technical staff along with the lack of efficient educational strategies for training project managers contribute to the observed scenario of inadequate management (Mandl-Striegnitz, Lichter, 1998). Partly due to this educational gap, novice managers usually take “fire-fighting” actions based on intuitive decisions, while experienced managers apply lessons learned from past projects.

By analyzing past situations and evaluating the different paths that a project could have followed if specific decisions were made, students can enhance their management skills. In this sense, learning approaches such as case studies and role-play analysis are useful to develop the students’ expertise and seem to play an important role in enhancing management effectiveness.

In this work, we present our experiences in developing and using simulation-based games for project management education. These experiences have shown us several difficulties related to game development, such as the lack of well-documented processes and integrated tools to support the organization of simulation-based games. According to such perception, we propose a high-level process and a model-based infrastructure for game development. Instead of using models only to describe game element’s behavior, we use models to describe the underlying story and its graphical representation.

We organize the remainder of this paper as follows. Section 2 discusses the desired characteristics that should be provided by an educational approach concerned with project management training. Based on such characteristics, in Section 3, we present a model-based educational game for project management and discuss its experimental evaluation. Section 4 presents a game development process and infrastructure to support the design and execution of simulation-based educational games. Finally, in section 5, we draw some conclusions and present future work directions.

2. Towards a learning-by-doing approach for project management education

Educational models describe the major interactions among the instructor, students, supporting materials, and other teaching resources. Most current educational models are content-driven and instructor-centered: a teacher is responsible for selecting what the students will learn, when and how the learning process will be conducted (Prensky, 2001). On the other hand, some studies have shown that adults prefer to learn based on experience and they usually learn better what they can apply to solve their
current problems (Knowles, 1984). Experiencing faulty projects due to inadequate management, for instance, is one way of increasing adult’s motivation into a learning process (Drappa, Ludewig, 2000).

Regarding software development, large-scale software projects usually present dynamic complexity, being characterized by feedback loops, non-linear behavior, and cause-effect relationships distant in time (Sterman, 1992). Such complex behavior is not usually precisely predicted by human cognition and intuition. Thus, when a decision is required, the straightforward mental derivation of possessed knowledge into a solution is rarely reliable. Therefore, since project behavior is not easily derived from basic principles (the content to be learned), content-driven educational approaches seem to be more effective when complemented with techniques that exploit the practical usage of the content under study (learning-by-doing).

Simulation, through its ability to shorten the observation period between actions and their consequences, supports the learning process by reducing the time that would be required to analyze the element or process under interest in the real world. Even more, in many situations, the cost and risks associated to real and pilot projects may be unaffordable or undesirable. Finally, a simulation model can be customized to present the student with situations whose enactment in real projects would not be possible due to practical constraints.

Although the efficiency of simulation in education is yet to be properly established (Größler, 2000, Spector, Davidsen, 1998), some empirical studies (Pfahl et alii, 2003, Maier, Strohecker, 1996) indicate that a business simulation approach with role-playing scenarios seems to be useful in educational contexts. However, some limitations of such strategy are well known. For instance, a simulation is usually conducted by setting up parameters and observing the behavior of a system without intervention. This unobtrusive observation rarely occurs in the real world, since agents may interfere when the process under interest does not behave as expected. Moreover, it may be difficult for an inexperienced trainee to trace and analyze observed simulation results to real world behavior, compromising the motivation and the practical construction of knowledge, important educational goals for an adult learning-by-doing approach.

Therefore, a more active participation and control during the simulation should be allowed. In addition, we should look for better ways to present simulation results, thus supporting the interpretation of project behavior and cause-effect relationships. A better learning instrument for project management education would then require an environment where students can act as managers. For instance, a game look-and-
feel can be integrated to simulation models, enhancing the latter with motivational features, such as fun, challenge, attractive visual effects and a more compelling interaction model for students. With an active and flexible participation, the student feels himself in charge of the situation, exploring its contents and seeking clear defined goals, following his own agenda to overcome the challenges imposed by the game. Usability, fidelity, multimedia feedback and drama effects are also important issues to provide the virtual role-play experience to the player as a software project manager.

Due to these fine-grained requirements, the fast construction of new educational games for each different learning situation might be an infeasible investment with the currently available tools. By introducing the simulation model as a separate and flexible component composed by pluggable pieces of knowledge, different educational goals and situations can be performed by trainees reusing the same graphical machine and simulation engine. In the next session, we present, in details, the distinct game components and challenges to accomplish these requirements in the development of an educational simulation-based game.

3. The Incredible Manager game

To evaluate the game-based educational approach, we developed a simulation-based game, called The Incredible Manager\(^1\). Similar instruments for software engineering education can be found in the technical literature, such as the SimSE (Navarro, van der Hoek, 2005) and SESAM (Drappa, Ludewig, 2000) projects.

The Incredible Manager allows a trainee to act as a project manager, being responsible for planning, executing, and controlling a software project. While playing the game, the trainee’s goal is to complete a project whose cost and schedule are established during a planning phase and approved by stakeholders. Such approval phase allows the trainee to review plans that strongly underestimate or overestimate the resources required to complete the project. While planning and controlling the project, the trainee has to make several decisions, including team selection, allocation, effort dedicated to quality assurance, budget and schedule control.

---

\(^1\) The Incredible Manager game and other tools are available at http://reuse.cos.ufrj.br/riosim/tools.php/. Visited in Jan 10, 2006.
After having an approved project plan, the trainee can execute and control the project. Project execution occurs in continuous turns, consuming the resources requested in the project plan. The trainee must be aware of project behavior and take corrective actions when necessary. Visual effects and reports provide feedback, showing exhausted developers, late tasks, unsatisfied stakeholders, and so on.

The game is composed by three main components, as shown in Figure 1: the simulation model, the simulation engine, and the graphical machine. Each component will be discussed in the following sections.

3.1. The simulation model

The simulation model is an independent component that can be easily exchanged according to the educational goals. In fact, several simulation models can be provided to the graphical machine, each representing a game phase. Phases are independent and can be used to create multiple educational projects, with increasing complexity or addressing particular details. The simulation model that underlies the game is an extension of previous software development models (Abdel-Hamid, Madnick, 1991) and it describes the world and relevant aspects of the software project presented to the trainee. It is built using the System Dynamics (Forrester, 1961) metamodel notation (Barros, Werner, Travassos, 2002), an extension of the System Dynamics modeling language that encapsulates the complexity of the traditional constructors (stocks, rate, processes, and flows) in an object-oriented fashion.

The System Dynamics metamodel is composed by three distinct models: a domain model, an operational model, and scenario models. The domain model defines categories of elements (classes) that take part in a problem domain, their properties, behavior, and relationships. Properties are relevant numeric data about the class, while class behavior is denoted by a set of equations describing rules of how class instances react to changes in the model. The domain model does not describe a model for a specific problem, but for a knowledge area where modeling can be applied. It is a domain description, which must be specialized in operational models to describe a particular problem. While a domain model is composed...
by classes, an operational model is composed by class instances. An instance is a specific occurrence of a class in the situation being modeled. Each class instance may have different values for the properties defined in its class. Class level relationships define how class instances are linked to each other in a given model.

Scenario models, on the other hand, provide a library of generic management events and theories that can be integrated to an operational model. Senior managers can develop scenario models expressing experiences they have collected by participating in several projects, thus sharing their knowledge with less experienced managers. Currently, we have a library of about fifteen scenario models available for the simulator. These scenarios include theories regarding productivity and error generation rates due to experience, productivity due to learning the application domain, effects of overworking and exhaustion upon developers, communication overhead, and error propagation along the activities that compose a project network, among others.

### 3.2. The dynamic-structure simulator

The simulation engine is responsible for executing the models presented in section 3.1, iteratively calculating their equations to evaluate the behavior of game elements, such as characters, objects, and processes. In a project management game, a character may be a developer or a stakeholder; an object may be a set of use cases, a code module, or the project plan; a process may be the analysis activity, the design activity for a given set of requirements or an inspection meeting.

However, the simulation machine for a game must allow user interactivity during game execution. Most traditional System Dynamics simulators provide a static-structure simulation: although the behavior of domain elements changes over time, their structure, that is, the relationships among these elements, remain fixed from the first to the last simulation step. Usually, model properties are initially configured by an operator and the model behavior is evaluated after all simulation steps are executed.

This type of simulation is not well suited for games, where user interaction is a key issue. Such interaction describes player actions to control the project. Like in the real world, the manager should be able to hire new developers, ask them to work overtime or increase the effort on design inspections to improve product quality. In a project management game, these decisions may require changing the way in
which domain elements relate to each other (for instance, the developer assigned to accomplish an activity).

To address these limitations, we have developed a dynamic-structure compiler and simulator for the System Dynamics metamodel. The simulator translates operational and scenario models into traditional System Dynamics constructors (Barros, Werner, Travassos, 2002) and executes simulation steps continuously until the player performs an action (that is, interacts with the graphical machine). Upon receiving an action, the simulator performs the required structural changes in the operational model and translates the resulting model to the traditional constructors. An action may trigger the creation of new elements in the operational model, removal of existing elements, or modifications upon element properties. Once the structural changes are applied and the constructor-based model is generated, the following simulation steps will use the new set of equations, however without affecting the values calculated in simulation steps already executed. This process is transparent for the player and it is mediated by the graphical machine, which receives player actions and forwards them to the simulation engine.

3.3. The graphical machine

The graphical machine differs from a traditional simulator by presenting model behavior and its changes over time in a multimedia format. The trainees interact with the simulation process through an interface that resembles the real-world situation of managing a project: instead of handling with mathematical equations, the trainee deals with iconographic representations of developers, activities, and other elements that are usually present in a real software project. As shown in the qualitative evaluations that we conducted, this enforces motivation and enhances the learning process.

Figure 2 shows the game office environment. Color changes indicate the project network (in the top right), schedule and funds (in the bottom left) status. Developers’ status are shown through balloons and body expressions as they become idle, tired or in panic. The player can pause the game, ask the stakeholders to modify the project plan, visualize developer’s profiles, task assignments in the message area (in the bottom right corner of the screen), and open project reports (the blue bullets).
3.4. Qualitative evaluation

To evaluate the game-based approach for project management education, 24 people from 3 different groups were invited to run *The Incredible Manager* with a simulation model representing a software project. Seven participants from a management graduate class composed the first group. Eight participants from an industrial software development laboratory composed the second group, while nine participants from an undergraduate course composed the third. A total of 11 B.Sc. students and 13 M.Sc. / Ph.D. students ran the game.

While performing the study, participants were asked to play the role of a trainee using our game. The training session included a simulation (running the game) and a discussion session (presenting the lessons learned). All participants were asked to fill in two questionnaires. The first was filled in before the training section and contained questions about academic degree, personal experience and interest in software development and project management. The second questionnaire, filled in at the end of the training session, contained qualitative questions about the game-based educational approach and the game itself.

The results of the qualitative questions for the three different evaluations were very similar: (i) 100% of the participants approved the game-based model; (ii) it was observed that they learned the lessons presented; and (iii) it was observed that they increased their management skills. For 52.2% of participants, the training session was considered very pleasant. Finally, 87.5% of participants described that the game experience raised their interest in project management.
The remaining questionnaire results indicate that the game-based simulation learning is motivating, practical and fun for the participants. Challenge, visual effects and time pressure are also viewed as important factors for the engagement and entertainment during the activity. The participants, especially the novices, pointed out that the graphical feedback and the possibility of practical simulation of real project situations were very stimulating. The diversity of choices and the uncertainty from the model give a non-linear experience running the game in different situations, exploring several educational goals. Challenge, visual effects and time pressure are also viewed as important factors for the engagement and entertainment during the activity.

Although the evaluation results were positive, they cannot support the effectiveness of the game-based project management education. The focus of this evaluation was mainly to gain qualitative insights and future work directions. Unfortunately, measuring the effectiveness of an educational approach through quantitative data and hypothesis confirmation is a complex task. When conducting a research on the effectiveness of educational games, issues beyond the game should be considered, such as the user characteristics, the environment and the learning situation. There are actually many subjective and intervenient variables that can lead to biased empirical results, especially when comparing different approaches such as books, teachers, and simulators (e.g., teacher’s proficiency, book’s content, among others).

Moreover, our experiences pointed out that games should be adapted to different learning situation, allowing management concepts to be presented incrementally to trainees. The Incredible Manager game can support changes in the simulation model. However, changes in the user interface or in the story line behind the game would require a considerable effort on design and coding. This limitation highlights the need for a more agile game development process. We addressed this limitation by identifying common game components, defining models for each component, and building a machine that runs the game from its component’s models. In the next section, we describe each of these models and their application to game development.

4. Model support for simulation-based educational games

In a training game, the characters and objects with which the trainees can interact represent concepts of the knowledge area that is under exploration in the educational context. In simulation-based games, these
elements usually have their behavior dictated by a simulation model. To attend to the learning-by-doing tenets, these elements should also dynamically respond to changes in the environment, either internal or external, and to user commands. Moreover, educational games frequently use graphical effects to enrich their interactivity and fantasy, in order to capture user attention and motivation.

We identified three major aspects that should be addressed while developing a simulation-based game: a simulation component, a story that provides the context the learning process, and a graphical user interface. While the first component can be provided as a model, the two later components are usually developed as software modules (that is, they are designed and coded in some programming language). To improve our capability in developing simulation-based games, we propose a model-driven game development infrastructure that divides a game project into these three aspects. A distinct model maps each of these aspects:

- a behavioral model describes how the elements within a domain behave over time, according to story-level interactions;
- a story model maps the mathematical formulations presented in the behavioral model to elements pertaining to the game domain, such as objects and characters; and
- a graphical model depicts the elements composing the story model, presenting them to the user and controlling user interaction with the story model.

These models are integrated as a layered architecture, where the behavioral model represents the inner layer. A second layer manages the story model, commanding simulation steps upon the behavioral model, capturing results calculated by the simulator, and mapping such results to user recognizable states for characters and objects composing the story. Finally, the third and upper layer (closer to the trainee) handles user interactions, translating user actions to story level events, capturing story elements, and presenting them to the user.

The skills required to build the behavioral model (domain knowledge, mathematical modeling, among others) seem very distinct from the skills that are needed to develop the graphical model (visualization, spatial organization, drawing, etc). Thus, we assume that several persons will be involved in the development of an educational game. The separation of distinct game aspects in different models supports work distribution among the development team. To coordinate the work performed by these distinct groups, we propose a lightweight game development process.
Five steps compose the proposed process, as presented in Figure 3. The first corresponds to the elaboration of a textual description for the desired game, describing player’s goals, characters and objects involved in the game, environments where the story takes place, and states that can be assumed by game elements.

The three central activities of the proposed process have the objective of designing the aspects into which the game was divided: (i) simulation definitions; (ii) story definitions; and (iii) graphical definitions. First, the development team describes the simulation model. Next, they build the story model. Finally, the team builds the graphical model. Such models are addressed in detail in the following subsections. They can be built in parallel, but their integration requires the definition of interfaces among the distinct models. The game textual description supports these interfaces, providing a view of the desired game to be shared among developers.

The last process step regards the composition strategy. In this activity, the models built by the preceding activities are integrated and executed. In the following sections, we describe each model that composes a game.

4.1. The simulation model

The simulation model is the core of the game. It uses the same simulation engine that was developed for *The Incredible Manager* game. The model defines the types of elements that compose the domain addressed by the game and the relevant relationships among these elements. However, the elements that take part in a particular game are not defined in the simulation model, but in the story model. For instance, while the simulation model describes the generic behavior for software developers, the existence of a particular developer in a game, along with the developer’s characteristics, is denoted in the story model.
4.2. The story model

The story model is built atop the simulation model. It maps the abstract equations that compose the simulation model into real world concepts that are more easily recognizable by the trainee. To accomplish this mapping, the story model assigns descriptive attributes to each simulation model element, such as the name and curriculum of a developer, the details on how an activity should be developed, among others. Such attributes are not relevant for the simulation, that is, changes in their values do not imply in changes to the model behavior\(^2\), but they are helpful to facilitate the interactions between the trainee and the simulation model. Instead of handling with equations, the trainee interacts with elements that can be observed in the problem domain (a developer with a name, an activity with procedures for its execution, and so on).

As we have argued before, trainees in educational games are frequently adjusting the underlying model by making decisions that concern game elements. These decisions, such as changing the developer assigned to an activity or the number of hours that a developer is asked to work per day, change model values and structure, and thus have to be accounted for in the remaining simulation cycles. In order to provide such interactivity, the environment relies on the dynamic-structure simulator described in section 3.2.

The story model is responsible for mapping user interactions into structural changes in the simulation model. It associates a set of actions with each story element. User interactions can trigger these actions. Each action is described as a sequence of primitive structural operations that affect the underlying behavioral model. Such operations include the following effects:

- changing the value of an instance’s property, for instance, to update the project deadline;
- introducing a new domain element in the model, such as hiring a new developer or demanding the execution of a new activity;
- removing an element from the model, such as firing a developer;
- breaking a relationship between two elements, such as changing the activities’ execution order;
- building a relationship between two elements, such as allocating a developer to an activity.

\(^2\) In case they could affect model behavior, such attributes should be addressed in the simulation model and would not be considered descriptive attributes.
As in *The Incredible Manager* game, the simulation engine handles these operations by changing the simulation model structure, storing the values calculated in previous simulation cycles according to the former structure, and calculating future cycles with the new one.

A state machine controls the actions that can be enacted upon a domain element by the trainee in a given moment. It presents the relevant states for each domain element that can be perceived by the trainee. Each state is associated with a set of actions, and state transitions are triggered by changes in the simulation model variables. Transitions are defined as Boolean expressions based on the value of simulation variables.

Figure 4 shows an example of a state machine designed to control developer interactions for a project management game. It dictates how each developer will react to changes in the simulation model in consequence of user interactions. When the project starts, developers are available to execute any activity. They automatically pass to the “Idle” state when hired. After being assigned to some activity under execution, they pass to the “Working” state. Depending on the developer’s exhaustion level, its state can alternate among “Working”, “Tired” or “Exhausted”. Developers only return to the “Available” state if they leave the project team. Therefore, these three simulation variables determine a developer’s state: allocation to the project, exhaustion level, and association to one or more activities under execution. Other game elements, such as project activities, have distinct and independent state machines.

![Figure 4 - Developers’ state machine](image)

After defining new structures, like action and attributes, to be embedded in simulation classes, and state machines for each class, there is a setup step where the story modeler initializes all instances (domain class instances) that will take part in the game. For example, instances of *Activity* class are named *Analysis*, *Design*, *Inspection*, *Coding*, and *Testing*, all of them with distinct values for the *function points* property.
4.3. The graphical model

The graphical model contains visual representations for story elements. Decorative objects (such as desks, chairs, and pictures) are added to the game. Story level state machines are associated to presentation-level state machines, whose goal is to reflect visually the changes occurred in a story element. Therefore, story-level state transitions trigger actions in the graphical layer, such as moving or disposing an image. By using this strategy, it is possible, for instance, to distinguish a tired developer from a developer that is not exhausted: sometimes, the tired developer will be sleeping on the desk, asking for coffee, and so on. Graphical states are represented by animations (such as listening to music) or static images (such as thinking).

The graphical model also defines interaction environments (such as offices, laboratories, and rooms), which are linked by navigation paths associated to story objects. Finally, the graphical model maps user level interactions, such as mouse movements or the activation of mouse buttons upon the visual representation of a story element, into story level actions. This completes the cascade of user interactions, from mouse actions to changes in the behavioral model structure, through the three-layered game architecture.

4.4. Evaluating the Infrastructure

To evaluate the feasibility of the model based game construction infrastructure, we have developed a tool that receives the simulation, story, and graphical models, verifies their integrity, link their relationships, and executes the game. To test the prototype, along with the proposed models themselves, we have developed a project management game similar to *The Incredible Manager*. The simulation model from the original game was reused, but a story and a graphical model were built instead of about ten thousand lines of Java code that managed user interactions and story play in the original game. Some graphical constructors were redesigned and others were created to explore the multi-environment features provided by the model-based infrastructure.

In conventional game development approaches, a character’s behavior and its visual representation are usually spread along distinct (but related) parts of the game code structure. Moreover, our experiences with *The Incredible Manager* game have shown us that complex programming issues (such as memory
management, thread synchronization, and event handling) are frequently addressed during game construction. Given these constraints, advanced programming skills may be required for game developers.

On the other hand, in the model based game development approach, each proposed model encapsulates a distinct game design concern (behavior, context, and presentation), facilitating work distribution among a team with heterogeneous and complementary skills. Moreover, the game execution engine implements common computational aspects that are useful for several different games, thus freeing game developers from these specialized and error-prone activities.

Figure 5 shows a screenshot extracted during a game session for the project management developed using the proposed models and infrastructure. It represents the laboratory where developers work. The player can interact with developers and activities (the latter are shown in the board placed on the wall), change the current environment (by clicking in the door), and execute generic commands in a control panel (at the bottom of the screen). Computers, chairs, desks, printers, and telephones are just decorative objects.

In the illustrated scenario, the player is changing the developer assigned to an activity. Activity execution can be analyzed by looking at the board. In this project, the two parallel threads of activity execution represent two distinct use cases under development. Concluded activities are marked with “C” (analysis and design for both use cases). Bulls with triangles indicate on-going activities. Activities marked with the encircled square are disabled (in this case, inspections). Paused activities, represented by parallel rectangles inside a circle, are waiting for the conclusion of precedent tasks.
Some relevant variables are continuously displayed on the control panel during the game session. The right-hand area of the control panel shows the elapsed time since the project started and the budget consumed until this moment. These variables are chosen in the graphical model. The game ends when either the project is concluded successfully or allocated time/budget is consumed (implying project failure).

In the proposed approach, games are executed directly from models, establishing a well-defined interface among orthogonal game concerns: behavior, context, and presentation. Once a game is modeled, we believe that it should be easier to customize and adapt it to different learning situations, since constructors from separate models can be reused across different game projects. Finally, since there is no need for programming skills, education personnel without computer science background can be trained to build their own games.

5. Conclusions

Simulation-based games seem well suited to be introduced in a learning-by-doing education model, such as required by manager trainees. They give the students an opportunity to experiment the consequences of executing or neglecting important management functions, confront themselves with complex issues that must be resolved during project development, and test different approaches and solutions to project management problems. The lessons learned from decision-making and cause-effect discussions over their own projects become an important factor to improve and maintain the motivation and engagement necessary for a better artificial training situation for project management.

In this paper, we have presented our observations from applying an educational game to support teaching software project management. Moreover, our experience allowed us to define a three layered model infrastructure to support game development. One of the main benefits of this approach is that it allows designers to focus on the domain aspects, saving development time, which otherwise would be spent in programming. The game development infrastructure controls several aspects that are common to simulation-based games. The proposed story and graphical models are not dependent of a specific simulation model: several behavioral models for a given domain may share the same story model, allowing the preparation of distinct instructional lessons, each with a proper model addressing a particular issue.
Concerning future works over the proposed models for game-based simulation, there is a special demand for graphical tools for model construction and evolution. Model refinements and new scenarios should be developed to capture situations perceived in real software development projects. Since there must be a quantitative basis to these models, new empirical studies should be performed to gather this information. In addition, future research on software engineering game-based learning should involve multiple disciplines, such as pedagogical issues over the training environment and process, cognitive and motivational issues, including but not limited to usability, interaction, perception, and multimedia presentation of the game.

Acknowledgments

The authors would like to thank all the participants involved in the empirical studies and also CAPES and CNPq for their financial investment in this work.

References


