SDM – An Educational Game for Software Engineering

Troy C. Kohwalter Esteban W. G. Clua Leonardo G. P. Murta

Universidade Federal Fluminense, Instituto de Computação, Brazil

Figure 1: SDM

Abstract

Software Engineering is an area of computer science that focuses on practical aspects of the software production. The Undergraduate courses of Computer Science have disciplines of Software Engineering, but they are usually taught in a theoretic way and with only a few implementation exercises using the learned techniques and tools. A practical approach for the concepts studied during the Software Engineering classes would help the student in understanding the reason for using the presented concepts. Due to that, we introduce *Software Development Manager*, a novel simulation game where the player owns a software development company that counts with the help of a team, which is administered by the player, to develop products desired by customers. The purpose of this game is to assist in learning the knowledge of Software Engineering in a way that takes advantage of the benefits of fun and entertainment.

Keywords: games, software engineering, people management.

Authors' contact:

{tkohwalter, esteban, leomurta}@ic.uff.br

1. Introduction

Learning can and should be a fun process. Given this, one option to make learning fun is by using games with the aim to stimulate curiosity and provide motivation for learning. Prensky [2001b] gives twelve reasons for the fact that computer games are the most potentially attractive pastime in the history of humankind:

1. Games are a form of **fun**. That gives us ***enjoyment and pleasure***.
2. Games are a form of **play**. That gives us ***intense and passionate involvement***.
3. Games have **rules**. That gives us ***structure***.
4. Games have **goals**. That gives us ***motivation***.
5. Games are **interactive**. That gives us ***doing***.
6. Games are **adaptive**. That gives us ***flow***.
7. Games have **outcomes and feedback**. That gives us ***learning***.
8. Games have **win states**. That gives us ***ego gratification***.
9. Games have **conflict / competition / challenge / opposition**. That gives us ***adrenaline***.
10. Games have **problem solving**. That sparks our ***creativity***.
11. Games have **interaction**. That gives us ***social groups***.
12. Games have **representation and story**. That gives us ***emotion***.

There are other means that can provide some of these characteristics, for example, movies and books. However, games are the only process that conveys all these aspects at the same time [Prensky 2001b].

In the area of Software Engineering, the traditional teaching consists of lectures and some practical work with the intent of using the theory learned in class. However, these practical works are usually small and do not stimulate the student's interest.

The students do not show much motivation in these types of proposed work because they belong to another generation. This generation, called "Digital Natives" by Prensky [2001a], is used to receive lots of information in a short period of time, like to do parallel tasks, and prefer graphics or visual interactions instead of text explanations.

Prensky [2002] identified that one of the problems in learning is the lack of motivation from the student. To learn any subject it is necessary to make an effort, something that is rarely done without a reason. While the goal of educators is to convey the content, they do not bother to keep the student involved. Computer

games, on the other hand, have the main goal to keep the users involved and consequently make them want to devote more and more time playing.

Considering these facts, in this paper we present the Software Engineering concepts modeled in terms of gameplay and game mechanics theory, in order to present a possible game based approach for teaching and simulating Software Engineering. Based on our model, we describe how we created a new educational game focusing at teaching Software Engineering. The game, called Software Development Manager, or SDM for short, is primarily intended to assist in learning the concepts of people management, involving training, assigning tasks, and workload. These concepts are taught in theoretical lectures of Software Engineering and can be better assimilated using SDM, keeping the students interested in the subject through the fun offered by the game.

The SDM game conveys the importance of the management of the development team and each possible role that a member of the team can undertake. However, the most distinct factor of this game is the use of human characteristics to determine if the employees are qualified to perform certain roles.

This paper is organized as follows: Section 2 presents some related works relate to other education games for Software Engineering. Section 3 presents the modeling concept and approach of the SDM game. Section 4 presents the implementation of the game. Section 5 presents the experiments made to evaluate the game. Finally, Section 6 presents the conclusions of this work.

2. Related Work

In Software Engineering, students are exposed to various theoretical concepts and end up having little opportunity to apply all concepts learned in class. Due to this problem, some educational games were developed to address the concepts that are taught in theoretical lectures, as well as to assist in understanding the content.

In Navarro [2002], the authors created a simulation game of Software Engineering called SimSE. The purpose of this game is to address the gap in the traditional techniques of Software Engineering teaching where students are exposed to various concepts and theories, but have few opportunities to transform these ideas into practice.

In SimSE, the player assumes the position of a project manager who has a team of developers. As the player manages the software development, he can make hiring and firing decisions, monitor progress, assign task and buy tools. This game counts with a graphical interface that transmits all the necessary information to the player by dialogues or information windows. Figure 2 shows SimSE's graphical interface. The fundamental goal of the SimSE project is to allow the customization of the simulated process model and therefore to be used by professors during the presentation of content related to the software life cycle.

Despite having elements of staff management, SimSE uses a basic employee system consisting on the usage of two attributes, besides their name and developer role. One attribute represents software development experience and another coding knowledge.

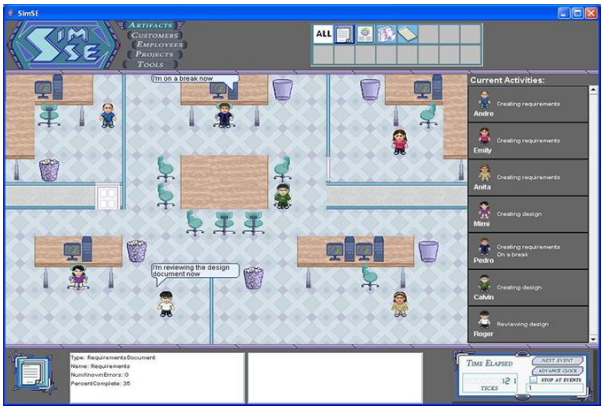

Figure 2: SimSE graphic interface

In Baker et al. [2003], the authors created a card game. The main focus of this card game, called Problems and Programmers, is the teaching of Software Engineering. Through a simulation of a software development process, from conception to completion, the players learn tactics to avoid problems during the development of the product.

In this game, players are project managers who work in the same company. The goal is to compete with each other in order to complete their projects in less time. However, players who adopt strategies to minimize the time or make risk decisions are affected negatively, while players who choose to follow the concepts of Software Engineering are rewarded. For this, the game depends on different types of card for each situation. Figure 3 show some of the cards in the game.

Despite showing important phases of software development, such as documentation, implementation, inspection and testing, the game Problems and Programmers lacks in some aspects, such as the card's structure and rules that make the game difficult to play with several people and impossible to be played alone.

In Figueiredo [2010], the authors created a card game called JEEES. The main focus of this card game is the teaching of Configuration Management [Estublier 2000] through a simulation of a project development environment.

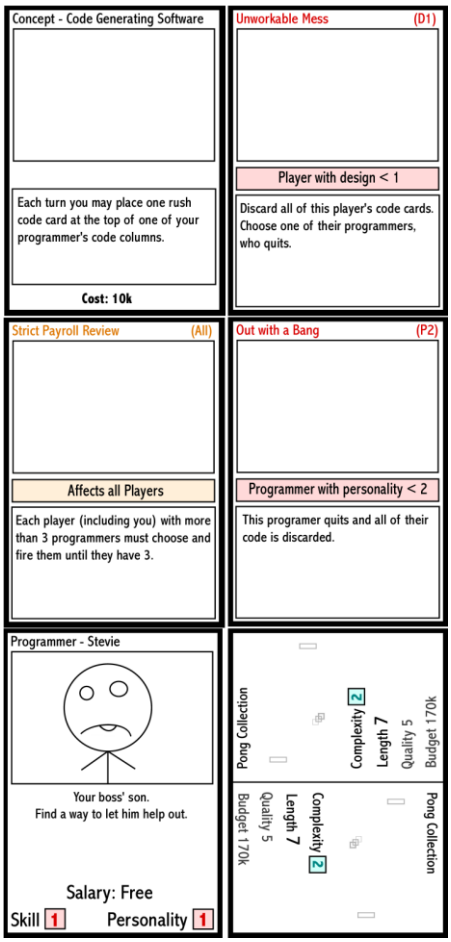

Figure 3: Problems and Programmers cards

In this game, players have the goal of finishing a software project through the completion and releases delivery. This can be accomplished by hiring developer teams as workforce to develop the software. The developer team card is illustrated in Figure 4. Despite being a game for Configuration Manager, JEEES can be adapted for other areas of Software Engineering with the addition of cards.

Figure 4: JEEE's developer team card

In JEEES, the management of the workforce is done by firing and hiring an entire developer team. There are only three attributes used to define the capability of the hired team, which represent the team's development ability, testing ability, and tendency to insert bugs in the project.

In Dantas et al. [2004], the authors created a simulation game for teaching Software Engineering named The incredible Manager, or in short TIM. The focus of this game is project management, where the player has the job of manager in a company. The player main tasks are to plan and to manage software development projects.

As project manager, the player establishes a development plan for the project and has the options of forming a development team to estimate the duration of each task, assign tasks to developers, make project plans, negotiate with stakeholders, control how many hours the team will work per day and determine the effort spent on quality assurance. All these options are available through the graphical interface, as shown in Figure 5.

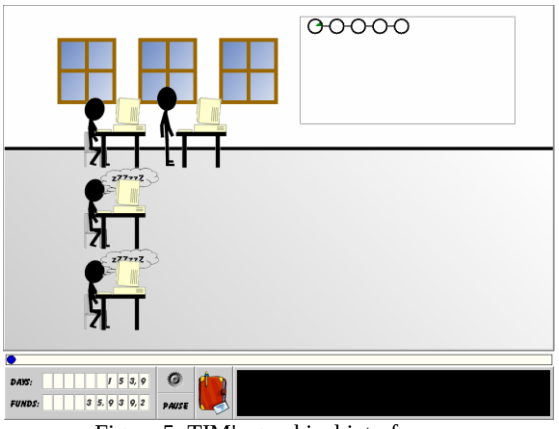

Figure 5: TIM's graphical interface

However, as in the SimSE game, it only presents two types of roles that can be performed during the development of the project: the manager, that is the player himself, and developers, that are the employees of the team. Nevertheless, these are not the only roles that are played in the development of software. There are others, which were not mentioned, but are equally important, such as system analysts and software architects.

While the mentioned works were modeled having in mind the Software Engineering concepts as a first requirement, there are other works that are mainly focused at the entertainment and gameplay aspects, such as Gamedev Story [2010]. Our work has a novel approach, since it tries to model the architecture having the fun theory in mind since the beginning of the concept, but also intending to stay very close to the Software Engineering theory and literature.

3. SDM Architecture

In the SDM game, proposed in this paper, the player has a team of employees who are used to develop software that are required by customers. The gameplay

and game mechanics are modeled presenting possibilities to the player to decide strategies for development and define the roles for each staff member. The software required by customers may have requirements that must be respected during development. From a gameplay point of view, these requirements help to balance the mechanics and rules. When the software is completed and delivered to the customer, there is a quality assessment of the software and a project completion payment.

Since SDM focus is people management, the main element of the game are employees, which represent the player's labor force. Since employees take a very important role, several existing features where expanded or added. These features include changes in possible roles that an employee can perform and the attributes used to calculate the employee's performance. Another element added is specialization, used to define the employee working competence. With the specialization system, it is possible for employees to undergo training to learn new skills. Also the concepts of working hours, morale, and stamina are used to modify the employee's productivity.

Figure 6 show a simplified version of SDM's class diagram focusing on the employee, showing his human attributes, types of specializations and the possibility of training to acquire specializations, and that the employee is affect by other employees that belong to the staff team. In small details it also illustrates the project and its characteristics and requirements that must be obeyed by the development staff.

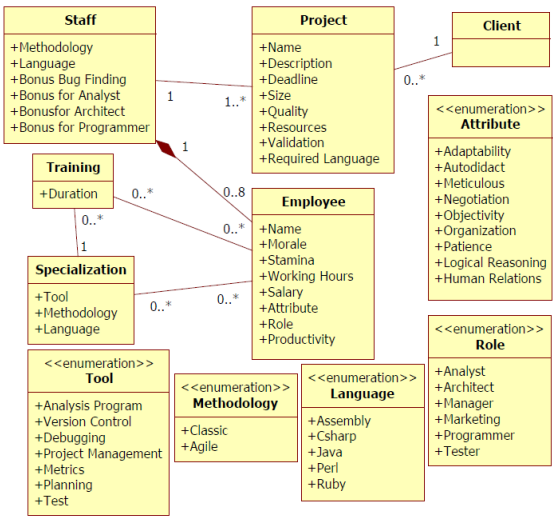

Figure 6: SDM's Simplified Class Diagram

3.1 Employees

In SDM, the employees represent the player's workforce. They perform planning tasks and develop the software through roles that are assigned by the player. In addition to the tasks that can be assigned, employees have certain characteristics that are used to determine the affinity to perform each possible task.

Through a dialog interaction, illustrated at Figure 7, it is possible to view the employee's profile, detailing all his characteristics. The profile is shown in Figure 8.

Figure 7: Dialog in SDM

Figure 8: Employee's sheet

3.2 Roles

The above-mentioned tasks are performed by the roles that an employee can play in the game. When a role is chosen for an employee, he will devote all his working time performing the desired function.

In the design of the SDM, six different types of roles were chosen and can be assigned to an employee. These roles are analyst, architect, manager, marketing, programmer, and tester. The purpose of using these six roles is to expand the scope of functions that can be performed by staff members. Each of these roles has a specific function that contributes to the development of software, which is described in the following.

The **systems analyst** serves as a translator of the user's needs into the architect's model that is used by the programmers during development. For this, the analyst must have a comprehensive knowledge of the business area in which the system is being developed in order to properly implement the business rules.

The **software architect** is the professional responsible for design and development of the system architecture. He is responsible for creating testing procedures to ensure the software quality is at the desired level, proposed by the customer. In addition, he is responsible for generating prototypes to validate requirements with the customer.

The **manager** is the individual responsible for planning and controlling the execution of the work of

team members. The manager, in the proposed approach, also manages the team's human resources and is responsible for hiring new employees. In addition to this task, the manager indirectly assists almost all functions performed by other roles during the project's development.

The role of **marketing** is responsible to negotiate the contract with the customer. The marketing can also have influence in the requests made by the customer to the system analyst, managing to convince the customer that an initially unwanted functionality is acceptable.

Codes do not generate alone. For this, it is needed **programmers** to develop the software. Therefore, the programmer is responsible for generating the software's source code. However, the programmer is also inadvertently responsible to put bugs in the software during development. These bugs affects the final quality of the project. The amount of bugs introduced is influenced by the programmer's experience with the programming language used and his capacity of working with the methodology adopted by the team.

Finally, the **tester** is the person responsible for carrying out the software verification to provide information about its quality. This responsibility includes the act of using the project in order to find bugs and correct them. As mentioned earlier, the tester works with the team's architect.

In the game, these roles are chosen by the player, as illustrated by Figure 9.

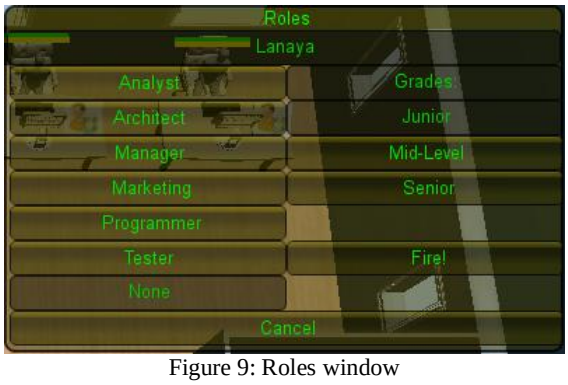

Figure 9: Roles window

3.3 Attributes

The employees have attributes that influence their job performance. However, these attributes are not directly related to each role, such as an attribute to be used for the role of manager and another for programmer. To make SDM more realistic and have a distinction of the other games, the system adopted for attributes is the usage of human characteristics of a person. These human characteristics, called human attributes in our work, are the attributes visible to the player so he can consult and figure out the performance an employee will have for a certain role. However, the mapping of these human attributes to the performance attributes is not informed to the player. With that, the player needs to use the common sense and make some experiments to be able to discover which attributes are more important for each role.

After performing a detailed study on employee's profiles [Santos 2005; Russo 2007] that plays certain roles in software development, we selected nine human attributes to represent the characteristics of the employee. The attributes are:

- **Adaptability**: the ability to react to changes. This attribute, specifically in our proposal, reflects the employee's ability to adapt to changes in scope and planning of the software. This attribute is widely used by analysts and architects and is also important for manager.
- **Autodidact**: responsible for the ability of learning without having an instructor. The individuals make their own research on the material needed for learning. This attribute is important for the programmer. Besides being used to determine the performance, this attribute is also applied when an employee is training, decreasing the time needed to complete the training.
- **Meticulous**: used to measure the employee's ability to pay attention to details of the problem, point to point checking the level of detail of the activities. This attribute is heavily used by testes and is also important for analysts and architects.
- **Negotiation**: used to determine the employee's ability to perform replacement persuasions, or persuade the other part showing the most relevant benefits of a point of view. This attribute, as its name implies, is essential during negotiations and is heavily used by marketing, but alone it does not determine the final performance of the negotiations.
- **Objectivity**: the ability to be objective. It serves to seek the simplest functional solution, especially during the software implementation. Therefore, it is a good attribute for programmers.
- **Organization**: the attribute responsible for the structuring of work and combination of individual efforts to make collective efforts. It is also useful for planning and creating schedules, thus making it a good attribute for managers.
- **Patience**: the virtue to maintain a balanced emotional control, without losing his mind over time. It consists on the tolerance to errors or unwanted events. The ability to endure discomfort and difficulties of every kind, from any time or anywhere, to persist in a difficult task. This attribute is important for testes, but no less important for other roles.

- **Logical Reasoning**: define the ability to seek a solution to a problem when data is available as a starting point, but no one is quite sure how to achieve the goal. A person with logical reasoning it is able to identify errors intuitively, understand the logic behind the problem and make mathematical calculations. This attribute is very important for programmers and testers.
- **Human Relations**: responsible for the interpersonal characteristic and communications skills of persuasion. Human relations help to explain ideas and to distinguish the technical vocabulary of the business. This attribute is heavily used by employees who perform the role of analysts, manager, and marketing.

The weight of the human attributes for the performance of each role proposed for this work is presented in Table 1. These weights are configurable.

Table 1: Human attributes weight for each role

	An	Ar	Ma	Mar	Prog	Tes
Adaptability	20%	25%	10%	5%	5%	5%
Autodidact	5%	5%	10%	5%	20%	5%
Meticulous	10%	15%	5%	5%	5%	25%
Negotiation	5%	5%	10%	25%	5%	5%
Objectivity	10%	10%	5%	5%	15%	10%
Organization	5%	10%	25%	5%	10%	10%
Patience	10%	10%	10%	20%	10%	15%
Logical Reasoning	10%	15%	5%	5%	25%	20%
Human Relations	25%	5%	20%	25%	5%	5%

An = Analyst Ar = Architect Ma = Manager
Mar = Marketing Prog = Programmer Tes = Tester

3.4 Specializations

Another important topic in our modeling is that each employee has specializations. These specializations can be from three types: programming languages, which reflects the language that the employee is able to work; techniques that are used to determine if the employee is familiar with certain methodologies adopted by the development team; and tools that are used to assist the employee to perform his tasks.

Programming language and techniques specializations are used to determine if the employee is able to work with the methodology and the programming language adopted by the staff, giving negative modifiers in case the employee don't meet the team's development setting. The tools specialization is used only to aid the employee, having no drawback for the lack of it.

3.5 Training

We propose a model where it is possible to train an employee so that he can acquire new specializations. The type of specialization to be trained can be chosen by the player at any time and have a duration of several days, influenced by the *autodidact* attribute. During the training period, the employee will devote his entire time on training and will not be allowed to exercise other functions simultaneously, such as working. Only when the training is completed the employee may resume his daily duties at the company.

With the possibility of training, the player can see that it is necessary, before starting the development or when adding a new member in the staff, to train his employees. If no training is made to meet the requirements of the project, the employee's performance will be reduced and the software quality will be negatively affected. Another learning that the player can get from the training aspect is that if an employee already belongs to the player's staff since older projects, it is very likely that he will need less and less training for new projects. Figure 10 shows all the possible skills an employee can be trained in the game.

Figure 10: Training Window

3.6 Working hours, morale, and stamina

The number of working hours is another factor that influences the performance of an employee and directly affects the employee's salary and his stamina. Morale and stamina models the employee's behavior and is directly related to their performance.

Stamina, as mentioned before, is affected by working hours. If the employee is doing overtime work, his stamina will start to decrease, thus causing a drop in his performance due to exhaustion. On the other hand, if he is working less hours, his stamina will increase.

With this relationship of working hours and stamina, it is possible to conclude that when an employee is working overtime, his productivity will be higher in a short term and then will begin to drop because of his exhaustion. After a few days, his overtime productivity will be negated by his exhaustion and if it continues, his productivity will fall beyond his normal productivity levels.

Morale also affects the employee performance, although not by fatigue but by the will to work. Morale

is responsible for the employee's desire to stay in business. If his morale is low, the employee may resign voluntarily from the team. Failure in payment is an example that affects negatively the morale.

This system of morale, stamina, and working hours provide the player other possibilities of strategic development of the software. It is up to the player to balance the benefits and drawback of using such aspects. Figure 11 shows a summary of each staff member which contain their daily salary and their levels of morale and stamina. Morale and stamina are also visually illustrated as bars that vary with size and color and floating letters that appears when changes occur, which are illustrated at Figure 12.

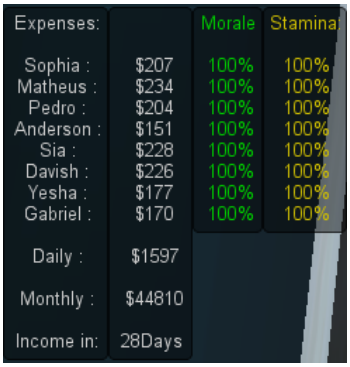

Figure 11: Staff summary window

Figure 12: Employee's working. Show changes on Stamina due to different working hours

3.7 Hiring

The player's staff can be changed during the game by actions taken by employees or the player's choice. The first alternative has already been explained, which is affected by morale. The second alternative, which is made by the player, consists on hiring or firing one or more employees.

The player can fire employees at any time. However, to be able to make a change in the staff in order to add a new employee, it is necessary that at least one employee in the staff is playing the role of manager. Talking with the manager through a dialog, the player can request additions to the team if the staff is not at full capacity. Figure 13 shows the hiring window, which allows modifying the staff. The window is divided into sections, where the first row, from left to right, shows the possible candidates for hiring, the middle row shows their respective costs for hiring, and the last row shows the player's staff. When a candidate is selected, his profile appears at the right side of the hiring window.

Figure 13: SDM's hiring window

3.8 Prototyping

Just like hiring, it is possible to design prototypes to elucidate some specific requirements with the client. This is achieved during the development process via a dialog with the staff architect, if there is any. Prototypes, in SDM, are intended to aid the comprehension and understanding of what the customer wants from the requested software.

Figure 14 illustrates the prototyping window, which is accessed through an architect. There are three possible types of prototypes, with the only difference being their complexity.

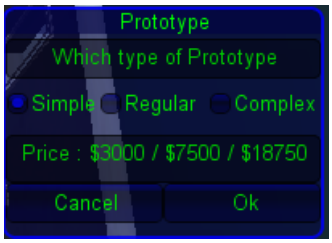

Figure 14: SDM's prototyping window

3.9 Negotiation

During the software development, it is possible to conduct negotiations with the client to change the project plan. These negotiations are only possible through a dialog with the marketing employee. The possible changes in the project plan are related to deadline, scope, quality, and resources. However, these changes need to be balanced, meaning that if one element is chosen, another one will need to be affected as well to compensate the changes. As an example, if it is requested more resources to the project, then the deadline will decrease.

Figure 15 shows the negotiation window, which have two fields: the upper field contains beneficial changes to the player's point of view and the lower field contains beneficial changes to the client. For a negotiation to be concluded, the player needs to pick one change from each field.

Figure 15: SDM's negotiation window

3.10 Leveling

During the course of the game, the player's staff gain experience points after developing projects. As the name imply, these points reflects in experience gained by participating in the project. When a certain number of cumulated experience points are reached, the employee gains a new level, resulting in an increase of attributes. However, the affected attributes during a level up depends on the roles he had between each level.

For example, if an employee works the entire period between levels as a programmer, then he may get an increase in all attributes, since all attributes are used for every role. But as a result of focusing on the programmer role, his autodidact and logical reasoning attributes will have a higher chance of increasing as well as the possibility of gaining multiple points in these attributes. This differential is measured by the time the employee spent on each role. So, if he had spent half the time as programmer and the other half as analyst, then his most affected attributes would be adaptability, autodidact, logical reasoning, and human relations. However, the increase in probability and in possible quantity would be inferior in comparison of focusing on only one role. Figure 16 is the window responsible for showing all the changes on the employee's attributes due to his current level.

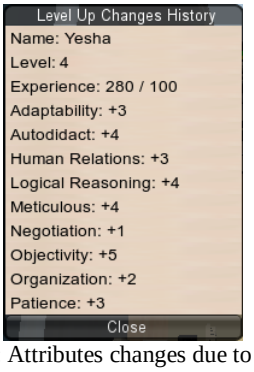

Figure 16: Attributes changes due to level

During the game, the player's company may also evolve, or devolve, according to the projects results. Initially, the company starts with just a few employees and can only take small projects. At the end of a project, the player company is rewarded with points that are used to calculate the company equivalent of level. This reward can be positive, in case the project was a success, or negative otherwise. As the company level increases, the complexity of allowed projects increase, resulting in better payment, but harder development. Due to the fact the company can lose points, it can also lose levels, resulting in restricting allowed projects. Figure 17 illustrates a completed project and the awarded experience points. The amount of experience is based on quality and complexity.

Figure 17: Project results

3.11 Differential

The SDM, in relation to the existing works, has many new aspects as a game. Employees now are an essential part of the development and as such they were expanded in terms of complexity.

In SDM the employees have human characteristics that determine their performance on the tasks, instead of a performance attribute directly related to each task. Besides, employees have specializations that affect their performance and make it necessary to do training in the initial phase of the development. They are also capable of gaining experience. Aside from attributes and specializations, the roles an employee can play also increased from two to six, adding new important roles that are used during the development of real projects. The player's company also evolves or devolves according to performance on previous projects. Lastly, SDM offers an infinite gameplay, always generating new projects to be taken by the player.

4. Implementation

The SDM game was developed using the game engine Unity3D (2010). Unity3D is a game development tool designed to let the user focus on creating games without worrying about game engine aspects. Figure 18 shows SDM inside the Unity game editor.

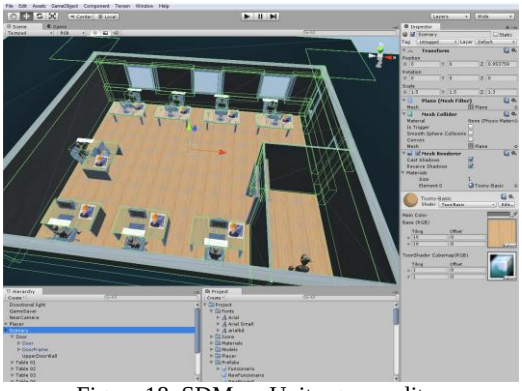

Figure 18: SDM on Unity game editor

In SDM, the player has the option to assign tasks and change the staff configuration by elements in the interface as well as view the project details. It is also possible to interact with employees through the usage of the avatar. In the game interface there is a time control mechanism, which in SDM is real time, allowing the player to pause the game or change game speed. Figure 19 shows the entire game interface, in which the upper elements are for staff management and lower elements for project details and game speed.

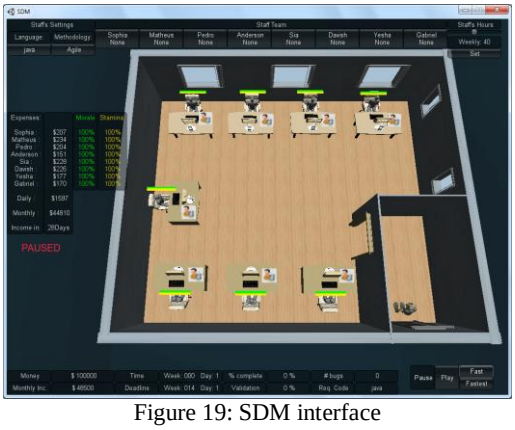

Figure 19: SDM interface

5. Experimental Evaluation

In order to evaluate the developed game, different students, with different profiles, were selected to play the SDM. The purpose of this experiment was to investigate what would be the student perception with the game and evaluate the contents of Software Engineering presented. For this, we exposed the selected people to a SDM gameplay session of thirty minutes and gave them a questionnaire, which was divided into two parts: participant characterization and game evaluation.

Among the selected twenty eight people, 92.6% are undergrad student, 3.7% master's students, and 3.7% PhD students. In this group, 18.5% has studied Software Engineering and 63% had no previously contact with Software Engineering. All evaluated participants enjoyed playing the game. Regarding the gameplay, 11% found it easy, 25.9% felt that the game was too complicated and the rest said it was normal in terms of gameplay. Referring to learning the contents of Software Engineering, 81.5% of the participants said they learned something new with the game. When asked if they would like to play the game again, 88.9% answered positively, explaining that it is was possible to apply the knowledge they learned about Software Engineering and the way the game handled the staff management. After having experienced the game, 85.2% said that the SDM sparked an interest in Software Engineering. The questions and data are presented in Table 2 and Table 3.

Table 2: Simplified Questionnaire

Question	Answers				
Education?	PhD	PhD student	Master's	Master's student	Undergrad student
SE experience?	Never	Read	Studying	Studied	Course
Liked playing?	Hated	Didn't like	Indifferent	Liked	Liked very much
Gameplay?	Incompre-hensible	Complicated	Normal	Easy	Very Easy
Learned anything new?	No	Yes	-	-	-
Want to play again?	No	Yes	-	-	-
Aroused Interest?	No	Yes	-	-	-

Table 3: Questionnaire's answers

Question	Answers				
Education	0%	3.7%	0%	3.7%	92.6%
SE experience?	63%	14.8%	3.7%	18.5%	0%
Liked playing?	0%	0%	3.7%	63%	33.3%
Gameplay?	0%	25.9%	63%	11%	0%
Learned anything new?	18.5%	81.5%	-	-	-
Want to play again?	11.1%	88.9%	-	-	-
Aroused Interest?	14.8%	85.2%	-	-	-

6. Conclusion

In this paper, we have shown a new proposal of modeling and implementation of an educational game for Software Engineering that can be used to aid the students to understand the various concepts that are taught on theoretical classes. We described our SDM game showing its aspects and differences on the other educational games presented in literature.

In our experiment, a game session in conjunction with a questionnaire was provided to the volunteers. The analysis of this experiment shows that the game helps the player to understand Software Engineering concepts that are shown in the game via an enjoying experience.

While developing this game, many other possibilities emerged, such as to include a system of affinity between the player's employees. This system would affect the integration of new employees in the team, causing the player to reflect on the decision to replace an employee in the middle of the software development. The new member would need, besides going through training, to meet the team's requirements and interact with other members to know how the team works on a daily basis.

A weak aspect of our game is that in its current version it is not possible to define iterations. Every day the player receives a feedback of the performance of each employee, but because of the huge number of information shown, the player may be disoriented in relation to assessing the pace of the project development. A work around to this problem is to allow the player to choose the size of iterations, in which he would receive information about the team's performance and each individual employee performance within the chosen period.

We consider that our goals where achieved with our current implementation of the game, as we analyzed the results of the experiments that shows the game as a enjoyable way to help students to understand some concepts of Software Engineering, especially about people management.

Acknowledgements

The authors would like to thank CNPq and FAPERJ for the financial support of this work. We are also grateful to all the volunteers that participated on the tests and validations.

References

Baker, A., Navarro, E.O. & Hoek, A. van der, 2003. Problems and Programmers: An Educational Software Engineering Card Game. In: ICSE, pp. 614-621.

Dantas, A., Barros, M. de O. & Werner, C.M.L., 2004. Treinamento Experimental com Jogos de Simulação para Gerentes de Projeto de Software. In: SBES.

Estublier, J., 2000. Software Configuration Management: a roadmap. In International Conference on Software Engineering. In: ICSE.

Figueiredo, K. et al., 2010. Jogo de Estratégia de Gerência de Configuração. In: III Fórum de Educação em Engenharia de Software.

Games, F., 2010. Game Dev Story. Available at: http://itunes.apple.com/us/app/game-dev-story/id396085661?mt=8 [Accessed May 5, 2011].

Higgins, T., 2010. UNITY: Game Development Tool. Available at: http://unity3d.com/ [Accessed May 5, 2011].

Navarro, E.O., 2002. SimSE: A Software Engineering Simulation Environment for Software Process Education. In: ICS.

Prensky, M., 2001a. Digital Natives Digital Immigrants. In: On the Horizon.

Prensky, M., 2001b. *Fun, Play and Games: What Makes Games Engaging*, In: Digital Game-Based Learning.

Prensky, M., 2002. The Motivation of Gameplay. In: On The Horizon.

Russo, R. de F.S.M., 2007. Tendência empreendedora do gerente. In: Gest. Prod., pp. 581-593.

Santos, S.C.G., 2005. Psicologia em Estudo - The information technology professionals and their personality analyzed by Rorschach technique. In: Psicol. estud.