PlayScrum - A Card Game to Learn the Scrum Agile Method

João M. Fernandes
Dep. Informática / CCTC
Universidade do Minho
Braga, Portugal

Sónia M. Sousa
Dep. Informática
Universidade do Minho
Braga, Portugal

Abstract— **To motivate and engage students and, consequently, improve the quality of learning, some researchers suggest new ways of teaching, including the use of serious games in the classroom. This paper describes PlayScrum, a new card game devised to allow university-level students to learn Scrum, an agile software development method. We present the card game, its rules, and how it supports the main concepts of the Scrum method. We also discuss how PlayScrum was validated, based on questionnaires filled in by master students who have played it.**

Keywords- software engineering; Scrum; card game; learning

I. INTRODUCTION

The software engineering process includes practices that promote the development of software systems and relies on methods to develop in systematic and organized ways robust software applications [8]. Agile methods are a subset of these methodologies, which propose approaches and principles different from those suggested by traditional methods, such as the "Waterfall Model". Traditional methods are based on a sequential development of activities. They are considered heavy because of its large bureaucracy and its weak adaptability to reality. Scrum is one of the most popular agile methods and involves little bureaucracy during the development process. It was designed for small teams (around 10 persons) and the software is produced incrementally based on the requirements collected throughout the development process [10-12].

Teaching software engineering requires addressing a very large spectrum of techniques and concepts. In particular, teaching software development methods proves difficult, since it is not easy to mimic in the classroom the realities faced by the software practitioners. The theoretical concepts may have been assimilated, but the student does not realize how and when to apply them in the "real" world. This lack of practical understanding by the students, even at the end of their academic education, does not satisfy the employers who have to hire them. To overcome this problem, new forms of learning that tend to give a more realistic view of the software engineering process have emerged recently. The teaching of software engineering using serious game is one of these alternatives. These games allow the simulation of the software engineering process in a more enjoyable and funny way, in which each student may play a role in the process. The games must be relatively easy to understand and interesting in order to attract students and improve the quality of learning.

This paper presents PlayScrum, a card game devised to help university-level students to learn the Scrum agile method. Students should learn concepts, practices and practical issues associated with the adoption of this software development method. This game is the result of a research about Scrum to objectively define the implication of its use in software processes and of the investigation of techniques and methods for creating games with rules that make their use feasible in a learning environment.

II. MOTIVATION

The use of traditional educational methods, like expositive lectures, is not always the most effective way to convey some concepts. In some contexts, students cannot get a real sense of what is transmitted to them because they do not understand the relevance of the teaching contents for their future profession. The use of serious games can be an effective approach to learn subjects, namely when feeling that business relevance is paramount. Several authors [1-7, 9, 13, 16, 17] have reported that, when learning with games, students can really acquire what is taught, putting aside their fears and anxieties and devoting themselves to the game. The concentration is intense and concepts are absorbed [13]. Software development is suited for being learnt with serious games, since many of its aspects are not technical.

Good software development planning is the key to producing good software. Scrum is used by companies with a leading position in the market, such as Google, Yahoo, Siemens, and British Telecom. Scrum is, therefore, seen as an important process for software development and was chosen to be the topic of the game described in this manuscript.

III. RELATED WORK

Over the last few years, several games were proposed as a method for learning software engineering [1-3, 5-7, 16]. Games, like SIMSE, SESAM, and Problems and Programmers, were created in order to improve the quality of the learning process in software engineering topics, like project management and software process and, so, to improve students' capabilities in making decisions. The designers found that the use of games as a complement to traditional

978-0-7695-3986-7/10 $26.00 © 2010 IEEE
DOI 10.1109/VS-GAMES.2010.24

teaching is much more efficient than as a single method of teaching [2, 5].

A. SIMSE

SimSE [7] is a simulation game played by a single player who must play the role of a project manager leading a team of developers. The player, among other things, can hire and fire employees, assign tasks to his team members, monitor the progress of the project, and acquire tools to improve the project. He can see all information needed to control and manage the project, namely related to his employees, to the progress of the tasks, to the satisfaction of the customers, to the budget and to the deadline for completing the project.

In the real world, software processes vary with the culture of the organization and its scope and, as such, SimSE is able to portray the differences that may exist in these processes.

B. SESAM

The SESAM project [3] led to the development of a simulator, in which a student can perform the role of a project manager. The player controls the simulator using a text interface, to read and write messages. He can hire or fire employees and ask them to perform tasks useful to software development, such as preparing the revision of the specification or testing the code. Most of the messages that the player receives are statements from the employees. The player is advised to carefully evaluate these statements, since they constitute the only source of information he has to make his decisions. The simulator has a number of internal variables, but these are not visible to the player, who only accesses the information that any project manager accesses in reality.

When the game ends, the player sees the obtained score and he can analyze his performance using the analysis tool provided by SESAM, which graphically displays the internal variables of the game.

C. Problems and Programmers

Problems and Programmers [1] is a competitive card game in which each student plays the role of a project manager. They have to lead the same project and the player who finishes first is the winner. However, to finish the project, competitors must be able to manage their budget, meet customer requirements and produce high quality software, among other things. Basically, they should follow the best practices of software engineering in order to avoid any obstacle.

The software development method followed in the game is the Waterfall Model and the software developed by the players goes through the different stages of this model (analysis, design, development, integration and test).

Players begin as such to create a column with analysis cards, then building a column of design cards at the right of the first. They continue by playing code cards to be grouped, at the end of the game, in an integration column. Thus, progress through different stages of the model is presented in a simple way and players can easily monitor their progress.

Even if the software process is fixed, the game gives players the freedom to define how to best achieve their objective [1]. While some players will place great emphasis on the analysis and design phases and carefully inspect the code developed before integrating it, others will rush all these stages in order to deliver the project as soon as possible. The game allows one to observe the strategic differences of the various competitors, which provides a powerful example of the consequences inherent in each of the strategies used in software engineering.

IV. SCRUM

Practices used in software development are constantly changing. Organizations have found that producing innovative products with high quality at a reduced price is often not enough to allow them to become the leaders in increasingly competitive markets. It is necessary to produce products quickly and provide flexibility in their development. In 1986 Takeuchi and Ikujiro presented a new approach to software development as a response to these new market needs [15]. While traditional approaches are based on a sequential development, this new approach is grounded on product development as a whole, made by a multi-disciplinary team that works together throughout the process. In 1993, Sutherland used a development method based on the study published by Takeuchi and Ikujiro, and he was the first to designate it as Scrum [14]. In 1995, Schwaber formalized the Scrum development process for the software industry with the publication of a paper at the OOPSLA conference [10].

Scrum is used to organize teams and promote a more productive software development, with higher quality. Its main idea is that software development involves environment variables that make the development unpredictable and complex, requiring flexibility to follow the changes [10]. Scrum enables development teams to choose the amount of work to do and how to do it, providing a flexible working environment. Scrum gives priority to work with more value for the customer, improving the usefulness of the delivered product, and increasing revenues generated through the provided software. Designed to adapt to changing customer requirements during the development process, Scrum is divided into sprints that have a typical duration of four weeks, allowing the definition of new requirements and adjusting the requirements to be developed in future sprints. Thus, developers can give customers what they really need, thereby increasing their degree of satisfaction.

Software development methods, like the Waterfall or the RUP (Rational Unified Process), provide the context and the definition of the process at the beginning of the project. In contrast, Scrum recognizes that the development process is never completely defined and uses control mechanisms to improve its flexibility. These mechanisms define that analysis, design, coding, and testing must be included in all sprints. A project that adopts Scrum starts with an abstract vision of the system to be developed. This view is based on the expected business results, but as the project advances, system requirements become increasingly clear and objective [11].

In general, projects that follow Scrum exhibit the following characteristics:

- Flexible deliveries determined by several environment variables.
- Small development teams composed of around ten persons.
- Interdisciplinary collaboration.

A. Phases of Scrum

Scrum is divided into three major phases during the execution of a project:

1) The Analysis phase consists in planning the new version of the system based on the requirements, to determine the time and cost of the implementation and, to define minutely how must be the requirements' implementation. This phase involves the definition of logical and physical architecture.
2) The Development phase is divided in sprints during which the functionalities of the system are developed, always considering the various environment variables that may affect the project.
3) The Closure prepares the system for final delivery to the customer. This preparation involves the system documentation, the testing and the delivery [10].

B. Scrum Control Methods

Due to its flexibility, Scrum requires control mechanisms to avoid that the system development falls into chaos using the following techniques to support the development:

- Establishment of lists that describe the requirements that were not properly developed. In those lists all enhancements requested by the customer are registered. This document also notes the expected outcomes of the system or the level of competitive advantage to be achieved as compared to its technological aspects;
- Definition of the product components that have to be changed to fit the requirements of the new system;
- Definition of the changes that must occur in the implementation of new requirements;
- Definition of technical problems that must be solved to implement the required changes;
- Definition of the risks that may affect the success of the project and produce responses to contradict them.

C. Scrum Actors

Projects, developed with Scrum, include three groups of stakeholders: the Client, the Team and the Scrum Master. The Client, representing the interests of those who ordered the system, introduces at the beginning of the project a list of requirements that must be implemented during the system development. This list defines the expected return on investment, as well as the delivery schedule. This document, known as the "Product Backlog", allows the Client to ensure that the most valuable features will be delivered first by the Team responsible for developing the system [11].

The Team is responsible for the technical development of the project. Teams have the power to self-manage, self-organize and are responsible to transform the "Product Backlog" provided by the Client in a sequence of tasks to be undertaken in iterations [11]. Teams define whose members are responsible for developing a task based on their expertise and personal will.

The Scrum Master is responsible for ensuring that all actors follow the rules and practices of Scrum, and he is bound for implementing the Scrum Method according the organizational culture, and for providing the expected results of the software development [11].

Members of these three groups form the actors that should not have more than ten elements. Other persons may have a stake in the results of the system but should not intervene in its development. Scrum ensures that the Team has sufficient authority to do what it finds necessary for the project to be successful.

D. Scrum Documents

In Scrum, one needs to register all requirements for the system as well as the priorities for their implementation.

The "Product Backlog" lists the requirements defined for the system. This document is prepared by the Client, who is responsible for its content, by prioritizing the requirements and the delivery system. This document is never complete, and as such, ends up being a mere tool to estimate requirements. The "Product Backlog" evolves as the product evolves. It is constantly changed in order to identify what the product must have to be a leader in the market and will be valid as long as the product exists.

The "Burndown Chart" shows the amount of work carried out at any time within the project [11]. This chart allows the correlation between the work done and the amount of work still to be developed by the Team to be visualized. This document allows the Client to assess accurately the implications of adding or removing features and also allows him to have an understanding of a possible gap between the reality of implementation and what was originally planned.

The "Sprint Backlog" is a document made by the Team to record the tasks to be carried out during the iteration. The Team compiles the initial list of tasks to be developed in the second part of the meeting, which indicates the beginning of the iteration. This document can only be changed by the Team, as it represents a clear picture of the work that it intends to develop during the iteration.

Scrum requires the Team to develop and to deliver new features at the end of each sprint. These features must be complete and ready to be presented to the client, who can use them immediately. This implies that the source code for the functionalities was reviewed, tested and is well structured. The documentation of the developed functionality should have been prepared and delivered to the Client.

E. Scrum Proocess

The Client is responsible for ensuring that the obtained system is actually the expected one and that it can maximize the return on investment. The Client prepares a project plan, the "Product Backlog", which includes a list of the functional and non-functional requirements for the project. These

requirements, when transformed into features, present the full scope of the expected product. The "Product Backlog" has a hierarchy of priorities and is divided into several deliveries. The priority of this list is the starting point of the project; however, other priorities often change during the project to reflect changes in business requirements and the speed with which the team can implement the requirements.

All work is divided into sprints that last thirty consecutive days. Iteration begins with a meeting in which the Client and the Team agree on the tasks to be developed in the current iteration. By selecting from the "Product Backlog" the requirements with the highest priorities, the Client tells the team what requirements are to be developed in this iteration. The team informs what percentage of these requirements can be effectively implemented in the iteration. Each of these meetings cannot exceed eight hours and is divided into two parts. The first four hours allow the Client to present the priority tasks to the Team that, in turn, can question him about the content and objectives to be achieved. The Team will select the requirements that can be developed and tested during the iteration. The remaining four hours will be used by the Team to plan their work during the iteration. The tasks that make up this plan are recorded in the "Sprint Backlog" and emerge as the iteration progresses. Early in the second half of the meeting, the iteration has begun and will have, from that moment, the maximum period of thirty days.

The Team meets daily for about fifteen minutes to review the state of the project. Each member of the team must answer three questions [11]:

- What did you do since the last meeting?
- What will you do today?
- What are the factors that can prevent you from fulfilling your goals for this iteration?

The purpose of this daily meeting is to synchronize the work of all members of the team and to schedule any meetings for the team to present their progress.

At the end of a sprint, there is a meeting in which the Team presents to the Client what was done during the iteration. This informal meeting permits the stakeholders to determine which options to be followed by the Team. After this meeting and before the one that starts the next iteration, the Scrum Master has a meeting with the Team to do a retrospective analysis of what happened in the previous iteration. For around three hours, the Scrum Master encourages the team to review its development process to make it more effective in the next iteration. All these meetings serve to verify the correct implementation of the practices advocated by Scrum.

This process is repeated throughout the duration of the project, knowing that as the project moves forward new requirements and priorities may be defined. New requirements that may arise in iteration cannot be developed immediately. The plan for iteration cannot be broken and, if new needs must be registered, they will be developed in future sprints through its priority level.

V. PLAYSCRUM

The card game PlayScrum, based on its predecessor Problems and Developers, is a competition game in which each student plays the role of a Scrum Master in a software development that follows the practices of Scrum.

PlayScrum can be played by between 2 and 5 players. The game is divided into sprints that differ from project to project and during which each player must develop a number of tasks defined at the start of the game.

PlayScrum includes the following elements:

- One board for each player.
- Product Backlog cards that determine the characteristics of the project.
- Problem Cards launched by opponents of a player.
- Concept Cards used by the player, as a remedy to problems launched by his opponents.
- Developer Cards corresponding to the development members of the Team of a player.
- Artefacts that represent the tasks performed by the Team during the project.
- A die.

The winner is the player who first performs all tasks defined for the project without errors or the player who has the highest percentage of tasks without errors, after the end of the last iteration.

A. The Board

We were able to identify certain shortcomings in the Problems and Developers game. One of them is the absence of a board for players to put their cards. As the participants were acquiring cards, it became very difficult to clearly identify which stage a particular card belongs to. As such, a board was conceived to facilitate playing with PlayScrum.

The board is an area in which players have their developers in columns and the acquired artefacts in rows. Artefacts are placed in the cells of the board, below the developer who acquired them. Artefacts acquired in this round are placed in the first line and artefacts acquired in previous rounds are placed in the second line. This separation exists because some problems cards differentiate on the rounds the artefacts were acquired.

Finally, with the board one can check more easily if a player does not exceed the budget set for the game, since the board allows players to have all their developers and concepts organized and visible to all participants.

B. The Product Backlog

Product Backlog Cards (Figure 1) provide all information of the project:

- Description: asserts the main features of the project. This description makes the game more realistic and helps the player to understand the system's main aim.
- Sprints: indicates the number of sprints on the project. A sprint lasts for 4 rounds. Thus, if a project has 7 sprints, then the game will be played in 28 rounds.

- Complexity: indicates the number of points that a developer has to spend to obtain a good artefact. The complexity of the project can be either 2 or 4.
- Tasks: determines the number of articles without errors that the user must complete to win the game.
- Budget: Money available to spend during the project. This attribute restricts the developers that can be hired as well as the use of certain concepts. Throughout the game, the value of the wages of all developers plus the cost of the concepts should not exceed the project budget.

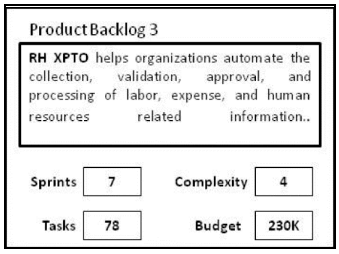


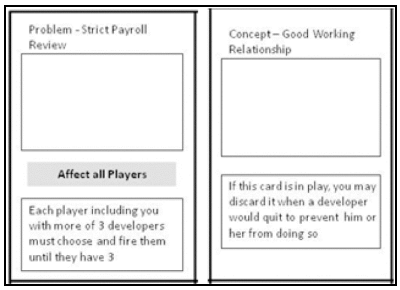
Figure 1. An example of a Product Backlog card.

C. *The Cards*

PlayScrum contains playing cards that make up the game. They are divided into four types: problems, concepts, developers and artefacts.

Problem cards (Figure 2a) describe classical problems that arise in the software projects, as well as problems that are not compliant with the practices imposed by Scrum. These cards are played by opposing players as obstacles to the progress of the development and have the following attributes:

- Definition of the problem: a short sentence that presents the problem to be handled. Thus, the student can see what problems may affect a software project developed with Scrum.
- Condition for the application of the problem: some problems can affect all developers, while others only affect developers with some characteristics. This difference depends on the problem and highlights the fact that there are external variables that can lead the project to failure.
- Consequence of the card: this attribute defines which penalty to be applied to the player who has received the card.



(a) (b)

Figure 2. Examples of (a) a Problem Card and (b) a Concept Card.

Problem cards are the main negative source of the game. They occur when players make bad decisions. These cards allow students to recognize the mistakes they made during the game and allow them to associate those mistakes with events that occur in the real world.

Concept cards (Figure 2b) describe the best practices in software engineering and can be used by players to react to the problem cards played by their opponents. The attributes of a concept card are:

- Definition of the Concept: a short sentence that presents the concept. The student can figure out what are the best practices to be followed when developing a project with Scrum.
- The consequence of the card: this attribute defines the positive effect of the card.
- The cost: some cards have a price.

Players use developer cards to progress in the game. Developers produce artefacts (code) that are needed to complete the project. These cards (Figure 3) have the following information:

- The **name** of the developer.
- A short **personal description** of the developer.
- The **salary** of the developer must be considered, since the sum of the developers' salaries and the cost of the concepts should not exceed the budget of the project.
- The **personality**, measured from 1 to 5, reflects the tendency of the developer to be a good employee, and is pertinent when receiving problem cards.
- The **skill**, measured in a scale from 1 to 5, reflects the knowledge level of the developer and determines the points that a developer possesses to perform the actions (buy, inspect and correct articles) during the game.

Figure 3. An example of a Developer Card.

Artefacts, which may or may not contain errors, correspond to the tasks performed by developers. There are two types of artefact cards:

- Blue artefacts (Figure 4) are considered good, because they have a 20% probability of containing a bug. Each blue artefact can be purchased by the value defined by the "complexity" attribute of the Product Backlog.
- Red artefacts in turn are seen as bad, because their probability to contain bugs is 60%. As such, each one can be purchased for half of the value set in the attribute "complexity" of the Product Backlog.

D. The Game

Before the game begins, a Product Backlog card is chosen. This card defines the number of sprints of the project and the number of tasks that should be done to end the game. The card also determines the project's complexity that determines the cost of the blue and the red artefacts, as well as the budget available to each player in the project.

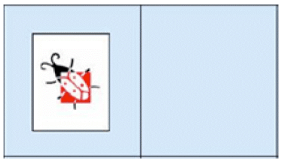


Figure 4. An example of an Artefact Card.

Each player receives 2 developer cards, which represents its Team. The sum of all developers and concept cards cannot exceed the budget set for the project.

As in the "Problems and Developers," the developer card determines the characteristics of the developer, namely salary, personality and skill. This last attribute implicitly defines the number of artefacts that the developer can produce.

Next, players mount their board and the cards are separated into four piles: one for developer cards, one for problem and concept cards, one for blue artefacts, and one for red artefacts. With the die, players determine who is the first playing and the game proceed in clockwise order.

At each turn, a player rolls the die and according to the number, draws cards from the piles:

- If the player draws 1, 2 or 3, only cards from the problems/concepts pile can be taken. He takes the number of cards indicated by the die.
- If the player draws 4, 5 or 6, he chooses 3 cards from the problems and concepts pile and the picks cards from the developers pile in a number equal to the number indicated by the die minus 3.

Problems and concepts stay in the player's hand until the moment he considered appropriate. Developer cards can also continue in his hand or can be placed immediately on the board, indicating the hiring of the developer. The player can have a maximum of six cards, including concepts, problems and developers. If the player has more than 6 cards, he must discard the excess cards before the end of his turn.

Artefacts are implemented according to the developers present at the board. The player may purchase as many artefacts as he can, as long as they are within the skill of its team of developers. An artefact represents a task defined by the product backlog and, when acquired, should be placed on the tray below the developer who implemented it. It is also necessary to separate artefacts implemented in this round or in previous rounds because some problems and concepts cards differentiate this situation. Artefacts implemented in this round should be placed at the forefront of the board below the developer who implemented them, while artefacts implemented in previous rounds must be placed in the last row.

In short, during a turn the player starts by rolling the die. Depending on the number shown by the die, he collects cards; he can hire developers to implement artefacts. In addition, during play, each developer can perform other tasks such as:

- Check his artefacts: the player turns one of the artefacts to see if it contains a bug. This task costs 1 point.
- Fix bugs in his artefacts: if a bug is found when inspecting an artefact, the developer can fix it by replacing the artefact with another one of the same type (blue or red). This task also costs 1 point.

Players can inspect and correct their artefacts during the sprint regardless of the round they were implemented. Similarly, problems and concepts apply to artefacts implemented throughout the sprint, except when the card itself invalidates this fact.

The player can also hire and lay off developers. However, to demonstrate the adaptation time that hiring a new developer implies this process must follow a strict chronological order. The player can fire and hire developers at the end of his move. As such, he dismisses the developer in a round and can only hire a new one in the next round. As the hiring is done at the end of the move, a new developer just starts to implement code in the next round. As such, the new developer will be one round without being able to produce.

At the end of his turn, a player is able to receive problems cards from his opponents. He can receive cards from the three players who played immediately before him.

The product backlog is divided into sprints that last for 4 rounds. Each player tries during these 4 rounds to get the maximum number of artefacts without errors depending on their developers and the problem cards that are used against him by his opponents.

At the end of 4 rounds, the moderator inspects the artefacts of each player and counts the number of artefacts without errors and the number of artefacts with errors (used to differentiate players in case of ties).

After this counting, the moderator removed the artefacts of each player and the process is repeated for subsequent sprints. Artefacts are placed back on the pill where they belong to. If at the end of iteration, a player achieves the number of tasks defined in the Product Backlog without errors, he is the winner. In case a tie occurs, the player who has fewer artefacts containing bugs wins. If the tie persists anyway, the winner is the player who had fewer bad artefacts during the project.

At the end of all sprints, if no player can reach the number of tasks imposed by the Product Backlog without errors, the player who is closer to that number wins the game.

E. PlayScrum vs Scrum

This section presents a relation between the Scrum method and the PlayScrum game, to make it clear how the latter addresses the practices and principles advocated by the former. Additionally, this exercise also exposes those aspects that were not included in the game and that should be considered in a future version.

- The **Product Backlog** is represented in PlayScrum by the card that has all the characteristics of the project.

- The **Sprint Backlog** is contemplated by the problem cards of PlayScrum.
- The **Burndown Chart** is not covered by PlayScrum.
- The game does not include the concept of **Prioritization of Tasks**. There are a number of tasks that must be accomplished to win the project, but nothing is said about their priorities.
- The concept of **Sprint** is considered by PlayScrum. All sprints have the same length (four rounds).
- The **Daily Meeting** is addressed by PlayScrum by the existence of problem and concept cards.
- The **Delivery of features 100% developed at the end of the iteration** is evident in PlayScrum, since at the end of iteration, the moderator collects all the artefacts developed by developers, and differentiate tasks with bugs and without them For the purpose of winning only the ones without errors count.
- PlayScrum enforces **Small teams,** since a player can never have a large number of developers, as the budget never allows it.
- **Flexibility and adaptability** are shown in the problems and concepts cards of PlayScrum.

The list presented above shows that PlayScrum includes almost all Scrum main principles and practices, namely the concept of sprints, the existence of the Product Backlog and the Sprint Backlog, small teams, and the delivery of features at the end of each iteration fully developed.

The points that were not included in PlayScrum are the prioritization of tasks, the Burndown Chart and the meetings that occur at the beginning and at the end of the sprints.

VI. VALIDATION

In order to perform a qualitative and exploratory validation of the game, thirteen master students at UMinho played the game and then were asked to submit written feedback in the form of answers to a questionnaire. In the future, we expect to conduct other evaluations experiments, namely by performing comparative studies between the aptitudes of students who played the game and those who did not.

Initially, students formed 8 groups to play the game. However, this number of participants was too large and at the end of the first sprint, the 4 groups with the worst scores were eliminated. Following this, they completed a questionnaire stating their thoughts and feelings about the game in general, their opinions about the pedagogical effectiveness of the game for issues on software engineering process, and their educational and professional background in software engineering. Some of these questions required a numerical answer on scale from 1 (low) to 5 (high), while others allowed students to give their responses in a free form.

A. Experimental Results

In general, students' feelings about the game were favourable. The first question *"Do you consider PlayScrum enjoyable to play?"* had an average score of 4.4. This result demonstrates very clearly that the 13 participants enjoyed their experience with PlayScrum. To reinforce this idea some students wrote:

"*The game is very interactive and allows players to think about the next moves in advance*".

"*The game is interesting and very enjoyable to play. It's a different way of seeing the problems in these areas*".

It also emphasized a negative point that implied to change the maximum number of players in PlayScrum from 8 to 5:

"*The main drawback I found is the time of the game that can be very long*".

The second question *"What is the level of difficulty of PlayScrum? What is the most difficult part of the game?"* had an average score of 2.5 that shows that the level of difficulty of the game may be considered appropriate. Regarding the most complex part of the game, some students also answered

"The most difficult is to manage the budget and decide what kind of artefacts I have to implement";

The third question *"Do you think PlayScrum increase your knowledge of Software Engineering?"* got an average score of 3.8. This score makes evident that students see PlayScrum as a good complement to classes. They also answered

"*It allows us to easily understand the agile methods concepts*".

"*The game can be a good complement to classes because it is a fun way to look at project management*".

Question 4 "*Do you gain new knowledge about Software Engineering with PlayScrum?*" received an average score of 3.3, which means that students see PlayScrum as a new way of learning Software Engineering but as a complement to lectures on classes.

The fifth question "*Does the game teach, in general, the software engineering process? Why?*" earned an average score of 3.6. The analysis of this result shows that students could better understand the process of software engineering by playing PlayScrum. To reinforce this idea some students wrote:

"*It helps to understand the concept of iteration and incremental development process in software engineering*".

"*It helps to realize the difficulty of producing software correctly, quickly and within budget*".

The sixth question "*Would you include PlayScrum in a software engineering course?*" obtained an average score of 3.8, which means that students agree to integrate PlayScrum in a software engineering course. Although this question does not require a textual answer some of the students choose to explain the marks given.

"Yes, because it is a practical application of what we learn in class";

Finally, the last question "*In general, what is your opinion of PlayScrum?"* aims to understand what should be changed in PlayScrum. This question can also define several possible directions for future work on the game.

Students were very cooperative in their responses and gave some suggestions:

"*The strengths of the game are working with the budget, developers and artefacts. The weakest point is that one rarely uses the possibility to 'turn' tasks and replace them*";

"*Linking the number of tasks and sprints with the number of players to prevent the game from becoming too long or even very fast*";

The dismissal of developers was also included in the rules of the game after this validation because it really makes sense to be able to exempt an element that does not meet the expected objectives.

During this validation, it was possible to realise that students also appreciated the fact that PlayScrum enables them to understand clearly the role of a Scrum Master in the project management. Questions, such as "*Is it better to get a good developer or two average ones?*", were discussed by students at the end of the game. The budget of projects forced students to understand the decisions that must be considered and taken in software development. The game also demonstrates the importance of developing from the onset good quality software. At the beginning of the game, students risked getting red artefacts to get twice the number of artefacts in the same number of rounds. However, after seeing the amount of artefacts with bugs they had, they quickly opted to prefer blue artefacts and only go for red ones when they could not choose the blue ones. This allows students to clearly understand that the quality of a feature is paramount and that delivered products cannot have failures. In fact, it is less costly to produce good software from the beginning than to produce poor-quality software to be developed again, which delays the product's delivery.

Whilst most of the answers were quite positive, it was noticeable that students with a poor knowledge of Scrum had much difficulty to understand the game and its dynamics. This point clearly demonstrates that the game should not be used in isolation. Instead, it should complement the theoretical concepts taught in class to take full advantage of it.

In short, the evaluation of the game was very positive and proved that the game conveys practices and procedures of Scrum and Software Engineering in a more comprehensively.

VII. CONCLUSIONS

PlayScrum represents a first attempt at using a physical card game to teach students about the Scrum agile method. PlayScrum addresses many of the weaknesses of more traditional learning approaches and brings additional benefits in the form of face-to-face learning and enjoyable play. We believe that, when used in conjunction with lectures and projects, PlayScrum allows students to gain a solid understanding of real world lessons that might otherwise have been poorly understood or overlooked altogether.

PlayScrum has a very visual nature, is simple and fun to play, allows for collaborative learning, and provides almost immediate feedback to players about the lessons to be learned. We feel that the game represents a good balance between our stated objectives.

The results of our experiments show that some PlayScrum cards can be improved. Problem and concept cards can be more explicit in their scope. As indicated before, some features of Scrum are not covered by the game, but we plan to include them in future versions.

An electronic version of PlayScrum would be a good way to continue this project. This version could be even more enjoyable to play and could have an area that allows the students to access all the theoretical contents of Scrum, thereby complementing the game.

Finally, in the future we plan to develop a more general version of the game, to allow students to learn all agile methods. This path requires an investigation of other agile methods and a comparison among them.

REFERENCES

[1] A. Baker, E. Navarro, and A. van der Hoek, "An experimental card game for teaching software engineering", J Sys Soft 75(1-2):3-16, 2005. DOI 10.1016/j.jss.2004.02.033.

[2] D. Carrington, A. Baker, and A. van der Hoek, "It's all in the game: teaching software process concepts", Proceedings of the 35th Frontiers in Education Conference, 2005. DOI 10.1109/FIE.2005.1612152.

[3] A. Drappa and L. Jochen, "Simulation in software engineering training", Proceedings of the 22nd International Conference on Software Engineering, 2000, pp. 199-208. DOI 10.1145/337180.337203.

[4] M. Ferrari, R. Taylor and K. VanLehn, "Adapting work simulations for schools", J Educ Comput Res 21(1):25-53, 1999.

[5] P. Mandell-Striegnitz, "How to successfully use software project simulation for educating software project managers", Proceedings of the 31st ASEE/IEEE Frontiers in Education Conference, 2001. DOI 10.1109/FIE.2001.963884.

[6] E. Navarro, A. Baker, and A. van der Hoek, "Teaching software engineering using simulation games", Proceedings of the International Western Simulation Multiconference, 2004.

[7] E. Navarro and A. van der Hoek, "Software Process Modeling for an Educational Software Engineering Simulation Game", Software Process Improv Pract 10(3): 311-325, 2005. DOI 10.1002/spip.232.

[8] R. S. Pressman, "Software engineering: a practitioner's approach", 4th. ed., McGraw-Hill, 1997.

[9] J. M. Randel, B. A. Morris, C. D. Wetzel, B. V. Whitehill "The effectiveness of games for educational purposes: a review of recent research", Simulation & Gaming 23(3):261-276, 1992. DOI 10.1177/1046878192233001.

[10] K. Schwaber, "Scrum development process", OOPSLA95 Business Object Design and Implementation workshop, 1995. DOI 10.1145/260094.260274.

[11] K. Schwaber, "Agile project management with Scrum", Microsoft Press, 2004.

[12] K. Schwaber and M. Beedle, "Agile software development with Scrum", Prentice-Hall, 2002.

[13] B. Shneiderman, "Designing for fun: how can we design user interfaces to be more fun?", interactions 11(5):48-50, 2004. DOI 10.1145/1015530.1015552.

[14] J. Sutherland, "Agile development: lessons learned from the first Scrum". Available at jeffsutherland.com.

[15] H. Takeuchi and I. Nonaka, "The new new product development game", Harvard Business Review 64(1):137-146 1986. DOI 10.1225/86116.

[16] G. Taran, "Using games in software engineering education to teach risk management", Proceedings of the 20th Conference on Software Engineering Education & Training, 2007, pp. 211-220.

[17] D. Thatcher and C. Donald, "Promoting learning through games and simulations", Simulation & Gaming 21(3):262-273, 1990. DOI 10.1177/1046878190213005