TEACHING SOFTWARE DEVELOPMENT BY MEANS OF A CLASSROOM GAME: THE SOFTWARE DEVELOPMENT GAME

Carlos Mario Zapata J.
Universidad Nacional de Colombia.
cmzapata@unalmed.edu.co

ABSTRACT

Software development is not only a matter of information technology teams: business stakeholders can be involved in this task at various stages. In addition, Software Engineering has been traditionally taught to people, both technical and non-technical, by means of regular and well-known methods, but some other teaching strategies have been left out—games, case studies, forums, and so forth—that could be applicable to Software Engineering. There are some examples of these alternative methods being employed in sciences like management, medicine, and law. However, for teaching Software Engineering, these strategies are still not applied. We propose in this paper the software development game, a strategy for teaching university students the dynamics of a software project. Also, we summarize the results of the application of the game to experimental subjects.

INTRODUCTION

Related to software development, business stakeholders are people who have some interest in a software application. Almost any subject or person can be recruited as a business stakeholder continuously during a given work task. This fact reinforces the need for teaching Software Engineering as a part of professional training for business people.

Software Engineering is a special discipline. It is a mix of three kinds of abilities: Engineering and Computer Sciences Knowledge, Software Development Methods, and Management and Communication Skills. The software industry requires professionals with the three abilities mentioned above, but Software Engineering has been taught by means of traditional methods (for example Lectures and "Toy" Practical Projects) with a small amount of changes in these methods (Baker, Navarro, & van der Hoek, 2005). Practical projects are also highlighted by Stiller and LeBlanc (2002), though some of the skills required by Software Engineers are not completely developed by using this method.

Wankat and Oreovicz (1993) proposed some methods for teaching Engineering in a broad sense, including lectures and practical projects; most of these proposed methods are rarely used in Software Engineering teaching. They suggested that traditional Engineering teaching must be complemented with new alternative methods, and they recognized four kinds of alternative methods for teaching Engineering:

- Technological Methods, for example computer or video games for teaching software design and other topics.
- Non-Technological Methods, including Case Study, fun activities without use of computers, and new strategies for practical projects.
- Labs, similar to Physics and Chemistry laboratories, as suggested by ACM and IEEE (2001) for Computer Sciences.
- One-to-one Education, for example in Question Answering and Tutoring.

Technological Methods are the most used alternative methods for teaching Software Engineering. However, there are still problems to be solved:

- They require a big amount of Hardware and Software resources. A typical class of computer games requires one computer for every student, and universities often have problems with the availability of this kind of resources.
- The growing number of registered Software Engineering students collides with the stable number of Software Engineering professors. The big amount of students, again, constrains the availability of human resources for teaching.

Software Engineering laboratories share with Technological Methods the first problem, while one-to-one Education share the second one with them.

Non-Technological methods are not constrained by the two mentioned problems. Fun activities and Case Studies can be hosted by only one professor, no matter how many students are registered in such course. Also, this kind of activities does not need Software and Hardware resources. However, non-Technological Methods have still low usage in Software Engineering teaching.

There are examples of educational games in sciences like Management, Law, and Medicine:

- "The beer game" (Senge, 1994) has been played by many generations of business managers, stock managers, supply department employees, students, and so forth. This game is a non-Technological stock management game.
- "Production of a maple leaf souvenir" (Wang, 2004) is a game designed by Manitoba University for teaching Total Quality Management concepts.
- "The Federal-mogul business game" (Petty, Hooker, & Barber, 2001) and other simulators (Al-Jibouri, Mawdesley, Scott, & Gribble, 2005) have been used for practicing project planning and controlling.
- Dynamic simulators and educational games (Foss & Eikaas, 2006) have been used for teaching engineering concepts in a broad sense.
- Case Studies have been one of the most used methods (especially in sciences like Law and Medicine) in some universities (Harvard, for example).

Related to Software Engineering, there are few reported experiences with this kind of games:

- Web-based games have been designed for promoting tournaments and competitions among students in the field of programming applications. The issues to be practiced are data structures (Lawrence, 2006), construction education (Kartam & Al-Reshaid, 2002), and general concepts about engineering (O'Brien, Bernold, & Akroyd, 1998).
- "Problems and Programmers" (Baker, *et al.*, 2005) is a card game for teaching software development life cycle.
- "Requirements game" (Zapata & Awad, 2007) is a game for writing documentation and building a little software piece in a 120-minute class.

We propose in this paper "The software development game", a non-Technological educational game for contributing to teaching some special topics related to software development. This game tries to solve part of the remaining problems of Software Engineering teaching.

This paper is organized as follows: in Section 2 we discuss the role of games as they are used in teaching some disciplines. "The software development game" is presented in Section 3. In Section 4 we summarize and discuss some of the results extracted from the application of the game. In Section 5 we present some conclusions, and in Section 6 we present future work related to this game and non-Technological Methods.

GAMES AS A LEARNING STRATEGY

Bohem (2006) suggests some strategies for future Software Engineering teaching like "Helping students learn how to learn, through state-of-the-art analyses, future-oriented educational games and exercises, and participation in research". This suggestion is supported by Felder, Woods, Stice, & Rugarcia (2000), who say that inwards-class learning must be active and collaborative; in this way, students are prone to react to knowledge process. In Lectures, students have a passive attitude, and professors are responsible for spreading the concepts.

Some authors argue that strategies like inwards-class games can help traditional Software Engineering teaching, despite their low usage in teaching. The main reasons for this assertion are:

- Motivation. Games are fun, and games can generate knowledge to their players. Lectures are not commonly fun, but games are almost undoubtedly enjoyable. Inwards-class games as a way to teach with motivation are discussed by Jensen (2006), Lee, Luchini, Michael, Norris, & Soloway (2004), and Dibona (2004).
- Representativity. We can simulate reality by means of inwards-class games (Kasvi, 2000). Gee (2003) exemplifies the kind of knowledge a child can acquire playing "Age of Mythology®"; in this case, past or imaginary stories are simulated in the game, and children can review similar concepts in ancient history.
- Interactivity and dynamism. Games are not only for representation purposes; we can interact with these representations (Kasvi, 2000). We can re-create entire battles or simulate desired behaviors in order to experience them over and over again. Video or computer games, or board games as "Monopoly®" or "Clue®", can be played uncountable times, and we can learn from them.
- Conflict. Challenges among players give interest to the game until the end (Pivec, Dziabenko, & Schinner, 2003).
- Safety. Through games is possible to re-create reality in a safely manner. No injuries or physical dangers are experienced in a game (Kasvi, 2000). Battles in "Age of Mythology®", for example, have a completely controlled environment, with no damages for players.

Klassen and Willoughby (2003) show other reasons for using inwards-class games as learning strategies:

- The gained insight should be unknown until the game is played. Players reach a meaningful learning stage, because they experience the game. New and important notions are deducted by students with this experience.
- Higher participation means higher learning. No matter if they win or lose, players with more interaction with the game have better chance to learn some lessons from the game.
- Low level of stress for the players. Results of the game are not important; participation in games is fun, and playfulness reduces stress.
- Simple materials can be used. An inwards-class game can be conducted with a board or a set of dices. With the exception of video or computer games, no technology is required for an inwards-class game.

For the above mentioned reasons, we propose an inwards-class game for surpassing some of the constraints identified in Software Engineering teaching. In the next section we explain "The Software Development game".

SOFTWARE DEVELOPMENT GAME

GOAL OF THE GAME

Players must build origami boxes with one of the following four groups of letters, SO, FT, WA or RE. Every box represents a software module (a part of a software piece that can be exchangeable with others). One group of four modules forms one software piece (a complete word, SOFTWARE, made of four modules). Every module must accomplish a set of pre-defined requirements which can be discussed—for the sake clarity—with the director of the game. The goal, therefore, is that the players must compete in groups to gain profits from an imaginary software company that makes software modules. In doing so, players are involved in the particularities of software development: communication breakdown, isolated work, and lack of planning.

GAME MATERIALS

- Specifications sheet. It has the origami instructions to form a gift box. The design is extracted from Glynn (1999) and is showed in Figure 1.

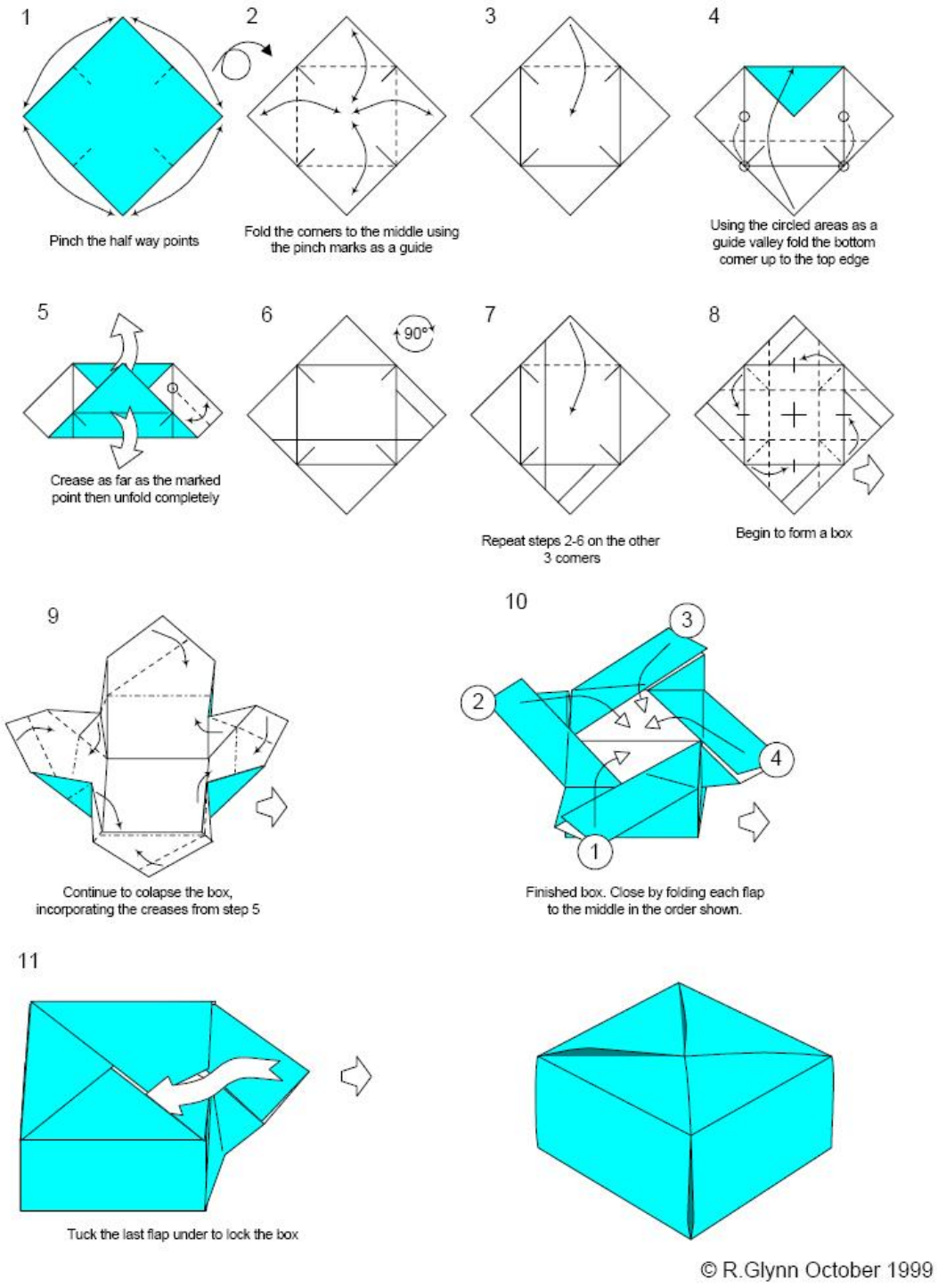

Figure 1. Specifications sheet.

- Raw materials sheet (for modules). It has four groups of letters (SO, FT, WA, and RE) arranged in a special way (see Figure 2).
- Game control sheet. It has a table for compiling data from the game, in order to determine the winner of the game (see Table 1).

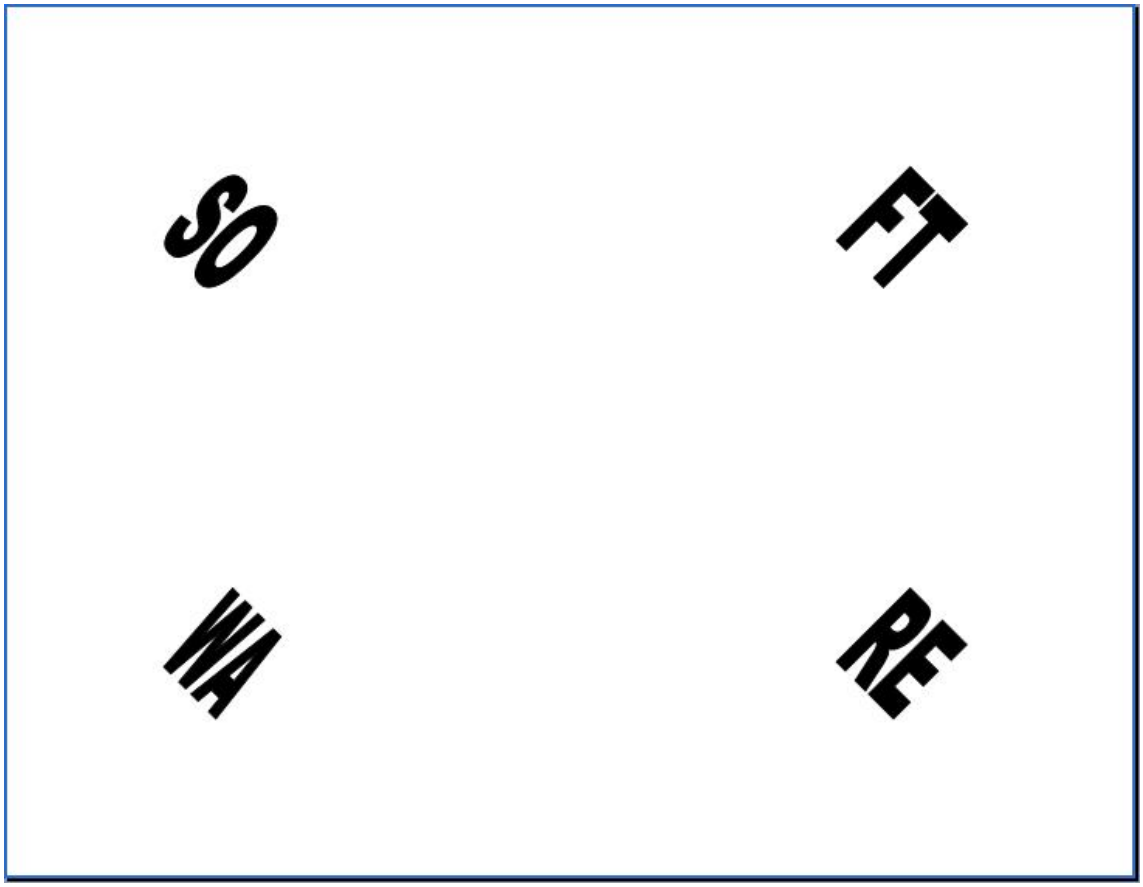

Figure 2. Raw materials sheet.

TEAM REGISTER SHEET

Team number

Description	Quantity	Unit Value	Total Value
Specification sheets		500	
Raw material sheets for modules		80	
Manager		300	
Workers		150	
Cost sum			
Software modules		150	
"Software" complete word		200	
Income sum			
Profits or losses			

Table 1. Game control sheet

PROCEDURE

- Players are distributed in teams. Every team has one manager and a variable number of operative employees.
- Game director explains the rules of the game in a 10-minute session. The costs and incomes from Table 1 and the importance of following the requirements must be included in this explanatory session.
- Game director "sells" to every team one specification sheet and the first raw materials sheet. In addition, game director

Number	Year/ semester	Course	Number of participants	Profile	Answered questions
1	2005_01	Introduction to Systems Engineering	59	First-semester students of Systems Engineering and Informatics	1, 2, and 3
2	2005_01	Requirements Engineering	12	Sixth to Ninth semester students of Systems Engineering and Informatics	1 and 2
3	2005_02	Introduction to Systems Engineering	43	First-semester students of Systems Engineering and Informatics	1 and 2
4	2005_02	Requirements Engineering	5	Sixth to Ninth semester students of Systems Engineering and Informatics	1 and 2
5	2005_02	English IV	10	Heterogeneous group of students, teachers, and university employees from different universities	1 and 2
6	2006_01	Introduction to Systems Engineering	90	First-semester students of Systems Engineering and Informatics	None
7	2006_01	Software Engineering and Linguistics Engineering	14	Mixed Group of students and teachers from these areas	1 and 2
8	2006_01	Advanced issues on Software Engineering	26	First-semester students from Systems Engineering Master course	1 and 3
9	2006_03	Introduction to Systems Engineering	60	First-semester students of Systems Engineering and Informatics	None

Table 2. Data of the participants

notifies every team the cost of manager and operative employees.

- Teams practice module building in a 5-minute session. Managers must decide how many raw material sheets need for a 50-minute session of the game. Simultaneously, game director must explicit the following requirements for the modules:
    - Unripped
    - Without wrinkles
    - Without draws
    - Invisible slashing
    - Letters within the central square
    - Cleanness
    - Without additional folding
    - Numbered in the right face
- Game director must "sell" the number of raw material sheets and pencils required by every manager. This process must be completed within 5 minutes. There is only one purchase per team. Surplus material has no refound process.
- Game director registers in a game control sheet data for every group: number of specifications sheets, number of raw material sheets, number of pencils, number of managers, and number of operative employees.
- Game director registers, in a visible place, the starting time for the 50-minute-game period.
- Teams build modules in the 50-minute period. Within this period only managers can talk to game director when asking questions. Game director must check the compiling of the rules.

Learning	%	Aspect
Speciality-task division	48	People
Previous planning of expenditures and limit time	45	Project
Team work	40	People
Communication among team members	28	People
Permanent analyst-stakeholder communication	21	Process
Final product must compile requirements	20	Product
Understanding the problem from the beginning	17	Process
Permanent monitoring of product quality	14	Process
"It is better low production with high quality than high production with low quality"	11	Product
Work under pressure	8	Project
Origami training	8	People
Acknowledgement of different stages of the development process	7	Process
"Training can be achieved through work practice"	6	People
"Previous experience is important in software development"	5	Project
A good strategy is hended	3	Project
External advising is important	1	Project

Table 3. Summary of answers for the first question.

- Game director announces the end of the 50-minute period. He must receive the elaborated modules matching the established requirements. The received modules and the number of software pieces are registered in the game control sheet. The role of the game director at this stage must be characterized by strictness in receiving the modules, in order to guarantee the needed understanding of the requirements.
- Game director summarizes the results and determines the winner of the game.
- Game director hosts a 20-minute discussion about game results and acquired learning. In this process, it is crucial that players must build the knowledge. Game director only assigns participation turns and registers opinions of the players on the board.
- Game director presents conclusions for the game in a 5-minute session.

OUTPUT OF THE GAME APPLICATION

Software development game has been played with nine different groups of people with different profiles. Table 2 summarizes application dates, name of the courses, amount of participants, and profiles of every group. Feedback has been obtained through a survey, conducted by the director of the game, in which participants must answer some of the three following questions:

(1) What did you learn from the game?
(2) What do you think you need to win the game?
(3) What kind of modifications do you think the game needs?

Table 2 also shows what question was answered by the groups.

The reason why the survey had open questions was that we wanted to know, avoiding biases in the answers, what the learning issues supplied by the game were. The survey was answered by 169 participants; question 3 was answered only by the groups 4 and 8, and question 2 was not answered by group 8. Due to the fact that group 8 was integrated by Magister students, this group showed a special behavior related to the game: they played the game with the purpose of improving the game itself.

Answers of the participants were tabulated for determining what it could be learning issues, requirements for winning the game, or improvements suggested by players of the game. Responses were grouped by similarities. Tables 3, 4, and 5 show the answers to the survey.

Due to the fact that Software Development Game is related to software project management, we analyzed the results with 4 P's model presented by Pressman (2004). The main aspects to be concerned in this model were: People, Product, Process, and Project. From this point of view, Pressman (2004) suggests that efficacy in a software project is highly influenced by People. This assertion was shared by the participants of the game, even more, three of the five common answers to the survey were related to People. These answers were:

- 48% of the participants learnt that the subdivision is very important for a software project into specialized tasks. The subdivision must be made by using the abilities of the people involved in the project. Software Development Game is a simulation of a real software project, and the roles are completely defined and distinguished as analyst, designer, programmer, project director, manager, and stakeholder.
- 40% of the participants learnt that team work is important for achieving the goals of the project.
- 28% of the participants learnt that good teams have good internal communication. Communication is a basic aspect of

Need for winning the game	%	Aspect
Ability-based task division	57	People
Adecuate planning of expenditure and limit time	48	Project
Acknowledgement and respect for the stakeholder requirements	34	Process
Good internal communication	27	People
Origami training and learning	25	People
A good director	22	People
Software development knowledge	21	Process
Permanent monitoring	15	Process
Quietness	10	Project
Motivation	8	People
Additional time	3	Process
Accurate tools (scissors, rulers, etc.)	3	Project
Trust in self capabilities	1	People

Table 4. Summary of answers for the second question.

Proposed modifications to the game	%
None	55
Previous training in origami techniques	11
Additional time for playing	11
Freedom for team group selection	9
Role exchange between people	6
Less manual job	5
Increasing of module size	4
Permissions to use better tools (scissors, rulers, etc.)	2
Diminishing of material costs	2
Variable time for analyzing the problem	2
Allowing of operative work of the manager	2

Table 5. Summary of answers for the third question.

a qualified software development team, and it is one of the most important elements in Software Engineering teaching.

Also, 45% of the participants learnt the importance of good planning of expenditures and limit times before starting software development. This assertion belongs to Project aspect of the 4 P's model, and it is highly co-related to People dimension. According to the participants, to make estimations of limit time and expenditures, it is necessary to recognize the capabilities of the team; this is crucial to define the real production capability of a software enterprise.

The last of the five main answers, as referred by 21% of the participants, was related to the Process aspect of the 4 P's model. People answer that the software projects need continuous analyst-stakeholder communication, and this issue is focused properly in the Requirements Engineering course; furthermore, it must be known and practiced by good Software Engineers, and it is crucial for determining stakeholders' needs and expectations.

The sixth lesson learnt, proposed by 20% of the participants, was related to the compiling of the requirements by the product itself, a matter of enormous importance for the acceptance of software by stakeholders. This is one of the final goals of Software Engineering, even though, it is the source of maintenance problems in the software, and it is the main cause of stakeholder's disagreement.

Other answers about Software Development Game learning are classified as:

- People: Training of the involved people in software development is important to achieve the goals of the project. Also, training can be the result of the development of many software projects. In addition, software process must be rapidly learnt by the development team. This is a challenge for Software Engineering teaching, and it is the justification for teaching software development methods (like RUP or XP) and the abilities required for the people to get adapted to them.
- Product: The participants mentioned the need for respecting software quality, no matter the required time for developing software. The pressure exercised by the stakeholders on the

proximity of limit time can originate avoiding software requirements, and this can be a source of problems after the software is delivered.

- Process: In this aspect, participants discussed three other issues. First, in software development, we must understand the problem from the beginning; a big amount of the game time must be expended in understanding software requirements, and this is one of the main goals of Requirements Engineering. Second, permanent monitoring by managers is needed in software development. Third, knowledge of development process is crucial in software development. The second and third issues are related to Implementation and Maintenance phases, and these phases are needed for a good quality of the software.
- Project: In this aspect, participants mentioned something about working under pressure, the need of previous experience, the adequacy of a good strategy, and the help of external advising, as supplementary issues to planning. In this sense, playing the Software Development Game helped the participants to identify some of the variables involved in real software development projects.

For the second question (what do you think you need to win the game?), participants answered consistently with the first question. Task division based on people abilities was the most common answer (57%), and limit time and expenditure planning was the second one (48%). Two responses not directly associated to the first question answers were:

- The need to have a good director for the project, expressed by 22% of the participants. This issue can be a summary of many capabilities previously mentioned, for example planning, and people hiring.
- Training and learning origami, mentioned by 25% of the participants. This issue can be compared with the required people capabilities in software development. For Software Engineering these capabilities are: analyzing, designing, and programming.

Notwithstanding the last question, related to the proposed modifications to the game, was answered only by 85 out of the 169 participants of the game, 55% of the answers suggested no modifications to the game. None of the proposed modifications was shared by a majority of the participants. With only 11% of the participants, two of the most voted modifications were:

- More training in origami before starting the game, and
- More time for playing the game (especially for making the boxes).

CONCLUSIONS

As a funny alternative to the traditional methods employed for teaching Software Engineering, in this paper we proposed an inwards-class game for teaching software development. This game is a simulation of a real software development project, and it does not need technological materials, making it easy to play in every environment.

Furthermore, we assessed the results of the game application with a survey to the participants. In this survey, we found that many of the issues concerned to the 4 P's model (People, Product, Process, and Project—a model for software management) were mentioned by the participants, mostly the need for planning, the subdivision of the development tasks, the team work, and the communication among participants in software development (including stakeholders).

The development of non-technological inwards-class games can be an opportunity for complementing the way that Software Engineering is traditionally taught. In this way, it is important to note that software development game does not try to replace traditional methods for teaching Software Engineering. Instead of this, we highlight the fact that software development game is only a complement for reinforcing what we can learn in Software Engineering courses. However, the practical way in what software development game is played makes it a good first activity for motivating the participation of business stakeholders in software development processes. A special course of Software Engineering for non-technical people can involve this kind of activities, especially in order to present, by means of a simulation activity, the consequences of bad performing stakeholder-tasks related to software development.

FUTURE WORK

Some work is still to be done about this topic:

- Development of non-technological inwards-class games about other issues like software development methods, consistency between diagrams, and modeling languages.
- Development of this type of games, trying to review the most important issues covered by this game, but trying to review them in-deep; these issues are planning, communication and people management.
- Improvement of the game by analyzing the suggestions made by the participants.
- Playing the game with people of a real enterprise, in other to validate the behavior adopted by the players against the roles of real software development projects.
- Development of an introductory course of Software Engineering directed to business stakeholders. The contents and the methodology must be adapted to the knowledge and skills related to non-technical people involved in software development processes.

AKNOWLEDGMENT

This work has been developed under the research Project "Un modelo de diálogo para la generación automática de especificaciones en UN-Lencep", funded by DIME.

REFERENCES

ACM & IEEE. (2001). Computing Curricula 2001 Computer Science. *ACM Journal of Educational Resources in Computing*, 1(3), 1–240.

Al-Jibouri, S., Mawdesley, M., Scott, D., & Gribble, S. (2005). The Use of a Simulation Model as a Game for Teaching Management of Projects in Construction. *International Journal of Engineering Education*, 21(6), 1195–1202.

Baker, A., Navarro, E., & van der Hoek, A. (2005). An experimental card game for teaching software engineering processes. *The Journal of Systems and Software*, 75, 3–16.

Boehm, B. (2006). *A View of 20th and 21st Century Software Engineering.* Proceedings of the 28th international conference on Software engineering, Shanghai, 12–29.

Dibona, Ch. (2004, February). A conversation with Will Harvey. *ACM Queue*, 21–27.

Felder, R., Woods, D., Stice, J., & Rugarcia, A. (2000). The future of Engineering Education II: Teaching methods that work. *Chemical Engineering Education*, 34(1), 26–39.

Foss, B. & Eikaas, T. (2006). Game play in Engineering Education—Concept and Experimental Results. *International Journal of Engineering Education*, 22(5), 1043–1052.

Gee, J. (2003). What Video Games Have to Teach Us About Learning and Literacy. *ACM Computers in Entertainment*, 1(1), 1–4.

Glynn, R. (1999). *Origami Gift box*. Retrieved December 15, 2008, from http://dev.origami.com/images_pdf/giftbox.pdf.

Jensen, B. (2006). Responding to the enrollment crisis—Alternative strategies to increasing students interest in Computer Science. *Journal of Computing Sciences in Colleges*, 21(4), 8–8.

Kartam, N. & Al-Reshaid, K. (2002). Design and Implementation of Web-based Multimedia Techniques for Construction Education. *International Journal of Engineering Education*, 18(6), 682–696.

Kasvi, J. (2000). Not Just Fun and Games—Internet Games as a Training Medium. In P. Kymäläinen & L. C. Seppänen (Eds.), *Cosiga—Learning With Computerised Simulation Games* (pp. 22–33). Helsinki, Skidoo.

Klassen, K. & Willoughby, K. (2003). In-Class Simulation Games: Assessing Student Learning. *Journal of Information Technology Education*, 2, 1–13.

Lawrence, R. (2006). Teaching data structures using competitive games. *IEEE Transactions on Education*, 49(1), 459–466.

Lee, J., Luchini, K., Michael, B., Norris, C. & Soloway, E. (2004). *More than just fun and games: assessing the value of educational video games in the classroom.* Proceedings of CHI '04 extended abstracts on Human factors in computing systems, Vienna, 1375–1378.

O'Brien, T., Bernold, L. & Akroyd, D. (1998). Myers-Briggs Type Indicator and Academic Achievement in Engineering Education. *International Journal of Engineering Education*, 14(5), 311–315.

Petty, D. J., Hooker, S. J., & Barber, K. D. (2001). The Federal-Mogul Business Game: The Development and Application of an Educational Aid for Planning and Control. *International Journal of Engineering Education*, 17(6), 546–557.

Pivec, M., Dziabenko, O. & Schinner, I. (2003). *Aspects of Game-Based Learning*. Proceedings of I-KNOW 03, the Third International Conference on Knowledge Management, Graz.

Pressman, R. (2004). *S. Software Engineering: A Practitioner's Approach, sixth edition*. New York:McGraw-Hill.

Senge, P. (1994). The Fifth Discipline: The Art and Practice of the Learning Organization. New York:Currency Doubleday.

Stiller, E. & LeBlanc, C. (2002). Effective Software Engineering Pedagogy. *Journal of Computing Sciences in Colleges*, 17(6), 124–134.

Wang, G. G. (2004). Bringing Games into the Classroom in Teaching Quality Control. *International Journal of Engineering Education*, 20(5), 678–689.

Wankat, P. C. & Oreovicz, F. S. (1993). *Teaching Engineering*. New York:McGraw-Hill.

Zapata, C. M. & Awad, G. (2007). Requirements Game: Teaching Software Project Management. *CLEI Electronic Journal*, 10(1). Retrieved from http://www.clei.cl/cleiej/paper.php?id=133

SERIOUS PLAY: SOFTWARE DEVELOPMENT GAME

Carlos Mario Zapata J.
Universidad Nacional de Colombia.
cmzapata@unalmed.edu.co

ABSTRACT

"Software Development Game" is a non-technological game for reinforcing some concepts about software development and project management. The game employs origami boxes for simulating these processes. The target audience is composed of any kind of professionals, who can be involved in any software development process along their professional activity.

DESCRIPTION OF THE GAME

People elsewhere can be involved anytime into a complex software development process. In fact, the development of software applications can be a common activity for any kind of enterprises. In particular, business organizations must deal with a dilemma: software applications are to be developed or bought? In either case, business stakeholders need some guidelines for acting in a proper way.

Traditionally, software development teaching has been an activity reserved for technical people, and business stakeholders—related to this activity—can be considered as non-technical people. Software development game is a funny way to simulate the environment of a software development process. With this game, technical and non-technical people can gain conscious about the importance of communication, requirements elicitation, and planning in software development process.

Players of software development game must "build" software modules (origami boxes) matching a set of previously stated requirements. In this process, communication plays a crucial role: requirements must be completely understood by managers in order to adequately transmit them to co-workers. Then, planning and internal communication can achieve success for one team.

Players are distributed in teams. Every team has one manager and a variable number of operative employees. The game director, who is also in charge of checking the compiling of the rules, gives every team one specification sheet (a map for cutting and folding sheets) and the first raw material sheet (a simple sheet with four drawn syllables: "SO", "FT", "WA" and "RE") and then, explains, in a 10-minute session, the rules of the game as follows.

First, teams practice module building in a 5-minute session. Managers must decide how many raw material sheets are needed for a 50-minute session of the game. Simultaneously, game director must write down on the board the following conditions of the modules at the moment of being hand-in: unripped, without wrinkles, without draws, with invisible slashing, with letters within the central square, clean, without additional folding, numbered on the right face.

Second, the game director should conduct logistic actions like "selling" the number of raw material sheets and pencils required by every manager in just one purchase moment in a 5-minute period, registering the costs presented in one form, and writing down, in a visible place, the game starting time. In third place, the teams build modules in the 50-minute period, in which only managers can receive insights about the game from the game director. After that, in order to determine the winner of the game, the director announces the end of the 50-minute period, receives the elaborated modules matching the established requirements, and fills out the incomes of the same previous form. Surplus material has no refund process.

Finally, the game director, assigning participation turns and registering the opinions of the players on the board, hosts a 20-minute discussion about game results and acquired learning, and presents conclusions about the game in a 5-minute session.

AKNOWLEDGMENT

This work has been developed under the research Project "Un modelo de diálogo para la generación automática de especificaciones en UN-Lencep", funded by DIME.