It's All in the Game: Teaching Software Process Concepts

David Carrington [1], Alex Baker [2], and André van der Hoek [3]

***Abstract* - A major challenge in teaching software engineering to undergraduates is that most students have limited industry experience, so the problems addressed are unknown and hence unappreciated. Issues of scope prevent a realistic software engineering experience, and students often graduate with a simplistic view of software engineering's challenges.**

Problems and Programmers (PnP) is a competitive, physical card game that simulates the software engineering process from requirements specification to product delivery. Deliverables are abstracted, allowing a focus on process issues and for lessons to be learned in a relatively short time. The rules are easy to understand and the game's physical nature allows for face-to-face interaction between players.

The game's developers have described PnP in previous publications, but this paper reports the game's use within a larger educational scheme. Students learn and play PnP, and then are required to create a software requirements specification based on the game. Finally, students reflect on the game's strengths and weaknesses and their experiences in an individual essay. The paper discusses this approach, students' experiences and overall outcomes, and offers an independent, critical look at the game, its use, and potential improvements.

Index Terms – software process, software engineering education, software project simulation.

INTRODUCTION

For several years, the first author has taught a third year undergraduate course on the software process to about 80 students. These students are typically enrolled in either a four year software engineering degree or a three year bachelor of information technology degree. The challenge in teaching this course is that most of the enrolled students have little or no industry experience, so the problems addressed by the course are unknown and hence unappreciated. Guest lecturers from industry are invited to motivate the content, but most students still fail to understand why this material is important.

When the first author became aware of the Problems and Programmers (PnP) card game [2, 3] developed by the second and third authors, he decided to use it in the software process course. Although simulations of software projects [6, 7] had been considered in the past as a way to overcome students' lack of industry experience, there are several aspects of PnP that make it especially appealing.

Problems and Programmers is a competitive, physical card game that simulates the software engineering process from requirements specification to product delivery. Each participant plays the role of a project manager, competing to be the first to complete the project satisfactorily. Each participant hires programmers and decides on the order in which development tasks are undertaken. Both players are constrained by the project budget and client demands for product quality. Players interact by playing problem cards against their opponent. The abstract nature of the simulated project deliverables means that the software process is the focus during the game. The rules of the game are easy to understand and its physical nature allows for face-to-face interaction between players, as well as play in the classroom without the need for computers. A game typically takes 40-50 minutes to play. The variety within the game provides numerous scenarios to explore. For a full description of the game and its rules, please see [2, 3] and the web site www.problemsandprogrammers.com.

While the game itself is useful, it is intended to supplement the usual approaches taken in a software engineering course, not replace them. Often, too little attention is given to how a supplementary educational tool of this kind can be smoothly integrated into a software engineering course. The intention of this paper is thus to pass on information about how the usefulness of a software simulation game can extend beyond the game itself. An approach is described whereby the game is played by the students, but is also used as the basis for a software requirements specification document and a reflective essay.

Another problem that this paper seeks to address is the fact that most educational simulation games are described only by their developers, with little or no confirmation by other users. While the second and third authors designed the game, they were not directly involved in the described use or evaluation of the game. Thus, this paper provides an independent evaluation of the game and describes its role in the larger setting of the software engineering classroom.

The remainder of the paper is organized as follows: Section 2 discusses the context for using PnP for teaching software process concepts. Section 3 describes how the card game was used in multiple teaching activities, and Section 4 explains the consequences of this approach. Section 5 contains

[1] David Carrington, School of Inf. Tech. & Elec. Eng., The University of Queensland, St Lucia, QLD 4072, Australia, davec@itee.uq.edu.au
[2] Alex Baker, Department of Informatics, University of California, Irvine, Irvine, CA 92697-3425, USA, abaker@ics.uci.edu
[3] André van der Hoek, Department of Informatics, University of California, Irvine, Irvine, CA 92697-3425, USA, andre@ics.uci.edu

0-7803-8552-7/04/$20.00 © 2004 IEEE October 20 – 23, 2004, Savannah, GA
34th ASEE/IEEE Frontiers in Education Conference

reflections on our use of PnP. In Section 6, some alternative approaches to teaching the software process are considered and compared with the approach described in this paper. Section 7 concludes and considers possible future developments.

CONTEXT

The software process course is compulsory for the software engineering students, but elective for all others. In preceding courses, students have developed programming skills, as well as learning about software design and UML. Students will have also completed a data structures and algorithms course, a database course, and a computer logic and organization course. Students in the course normally take a software project course either in parallel or subsequently, so one course objective is to equip students with the necessary knowledge to successfully complete a software project.

The topics covered in the software process course include: software lifecycles and processes, software process improvement, requirements engineering, software quality, verification and validation including inspection techniques, and software maintenance. The course is run for a 13 week semester with two hours of lecture per week for the whole class plus each student attends a one hour small-group tutorial, each one consisting of about 25 students. The course structure requires students to work in teams of three or four on three major assignments, and then complete a final individual essay.

The challenge in teaching software engineering to undergraduates is that most students have little or no experience with the typical problems in industry. While students do develop software as part of their university courses, their products are generally quite unrealistic in size, complexity and lifetime. Student assignments by necessity have a lifetime of weeks or possibly months, rather than years. It can be hard to motivate "best practice" in software engineering when students do not understand what the problems are.

Simulation is often used to build experience in other educational fields. It can be very motivating in an educational setting where real-life experience is too costly or time-consuming to provide in an educational setting. The classic example is the use of flight simulators to train pilots [11]. Simulation has been used in software engineering education, but not widely. Several such approaches are described in Section 6.3.

INTERVENTION

The availability of the Problems and Programmers card game provided an opportunity to trial it as a teaching tool in the software process course. The first step was to provide an opportunity for the students to learn and play the game. It was decided that the small group tutorial classes would provide the best environment.

I. Tutorial Exercise

In preparation for the one hour tutorial class, students were provided with information about the game and a set of instructions for playing it. The original instructions supplied with the game are eleven pages so a revised set was developed using just four pages to encourage students to read them. The original instructions were also made available in case students were interested. The students were advised to read the instructions in advance to maximize the amount of time available for playing the game.

In the tutorial, students were organized into teams of two to play the game, a deviation from the conditions under which the game was originally tested. The intention was to encourage discussion about the game and the choices necessary to complete the simulated project successfully. So in a tutorial class of 24 students, six games proceeded in parallel.

Because many students failed to complete a game in their one hour tutorial, the following week's tutorials were used to ensure that students had an opportunity to play a second game. Students were also encouraged to play additional games in their own time to gain further familiarity with various aspects of the game. They were informed that the game would be the basis for the second assignment and a topic for the final (individual) assignment.

II. Software Requirements Specification Assignment

The second assignment in this course normally involves developing a software requirements specification (SRS) document. Since the first assignment is an inspection of an existing SRS, students are familiar with the structure and contents of this type of document. The goal of the second assignment is to give students experience constructing an SRS.

Following the use of the PnP game in two tutorials, the SRS assignment required each student team to create an SRS for a computer-based version of the PnP physical card game. The implementation was to be based on the instructions already provided, and was to allow either or both of the two players to be humans or computer simulations. The fact that the students had played the game previously gave them a unique perspective on this activity. They had an understanding of the basic substance of the system to be developed, but they were forced to focus on the challenge of codifying this in a formal manner.

The SRS was required to contain both a natural language description of the requirements and UML analysis models (use case model, sequence diagram for the main scenarios, and a UML analysis class diagram). Consistency within and between models was emphasized. Students were not required to proceed with design or implementation of their system, but prototyping to clarify requirements was encouraged.

III. Reflective Essay

The final assignment in the course is an individual essay that requires students to describe their understanding of the scope and significance of the software process. As part of the essay this year, students were required to discuss how a game like PnP could help or hinder student learning of the software process and its associated concepts. They were encouraged to identify the strengths and weaknesses of the PnP game, and to suggest ways in which the game could be improved to

increase its effectiveness as a tool to facilitate learning about the software process. This activity encouraged students to think critically about the game's rules and their experience with it. Assessment of the essay is based on:

- the collection, analysis and presentation of relevant information in an appropriate style, and
- the presentation of a personal viewpoint, supported by external sources.

OUTCOMES

In this section, we review the outcomes for each use of the PnP card game in the course.

I. Tutorial Exercise

Although students were asked to read the instructions for the PnP game before their first tutorial, it was obvious that most of them had not done so. The games took quite a while to get started as there were many questions that were answered by the instructions. During the first tutorials, the focus was primarily on the game mechanics and rules. Consequently, many games were unfinished at the end of the hour and had to be abandoned as the card decks were required for the next tutorial.

The second week's tutorials were more successful with almost all games completed within the hour. This time, students were more engaged, and the discussion had moved from the rules of the game to the possible strategies for winning.

II. Software Requirements Specification Assignment

The specification assignment provided students with a second opportunity to consider the PnP game, but from a different perspective. The challenge of taking the existing physical card game and translating it into a computer-based game was significant and more difficult than originally anticipated. Although most students are quite familiar with computer-based implementations of card games (for example, Patience and Hearts), many student teams struggled to develop a coherent SRS (and associated UML models) and a consistent view of their desired game. About half the student teams used some form of prototyping to explore user interface options for their game. Those that did so generally had more consistent and realistic descriptions of the interactions between the game and its players. Finding an effective level of abstraction for the SRS description proved to be another major challenge.

III. Reflective Essay

The final essays are typically 10-15 pages in length with the sections discussing the Problems and Programmers game varying from 20 to 50%. Students raised a wide range of issues about the PnP game in their reflective essays, and it is difficult to summarize more than eighty points of view. We look at a selection of issues raised by students, illustrating them by quotes from their essays. As should be expected, students did not always share a common viewpoint.

Student attitudes to the overall game and how easy it is to learn and play were mixed, although most students rated the game as a positive element in the course.

- "Overall the game is effective at providing an abstract view of the benefits of following a process in order to complete a project, is very simple and easy to learn and is generally fun to play (there's something to be said for hitting your opponent with a problem card that makes all of his programmers quit)."
- "The game would, then, perhaps be good for use in an introductory level overview of the software process … The major downfall of this is that it takes a fair amount of time to become familiar with the game and the way it is played. … Ultimately the game itself is too complex to warrant the amount of time needed to master it, without significant returns to the time invested."
- "I must admit to disliking the game at first, it seemed to me to be a waste of time and effort, after playing it in tutorials and doing an assignment on the game I can honestly say that now I appreciate its simple yet effective use of the concepts of the software process."
- "In general I found the Problems and Programmers game to be a useful complementary learning tool to help to reinforce concepts from the course in an entertaining way. While the game is relatively simple by necessity, it can still teach a student valuable lessons about a wide range of concepts and possible problems associated with the software process. However the simplicity of the game can hide the common complexities encountered in software projects from students."
- "Very broadly speaking, Problems and Programmers would be a more effective teaching tool if it were more believable. Not that it is especially unbelievable at the moment, but rather, it lacks the power to convince. … I think the only way students will be truly convinced is through first hand experience, of both failure to adhere to the principles of good software engineering resulting in production of bad software or worse, and of proper use of software engineering techniques on a successful project."

One area where students did agree was the limitation of the PnP game being based on the waterfall life-cycle. This is not surprising when it is understood that this course spends considerable time describing alternatives to the waterfall model, so naturally students want to demonstrate their new knowledge. Indeed in their reflective essay, many students commented that the waterfall model was the only software lifecycle they knew prior to studying this course.

- "The first and foremost weakness that I see in the PnP game is that it focuses very heavily on the waterfall lifecycle model, and does not allow easy changes of the rules in order to incorporate other lifecycle models. … This is a problem because most students tend to be aware of the waterfall lifecycle model already, so the game does not extend their knowledge as much as it does enforce a stereotype about the software process that students are already quite familiar with."

Many students explored how the game might be changed to incorporate multiple lifecycles with most recognizing that such changes would be not trivial, for example:

- "Players would be able to select a lifecycle model to follow in order to deliver their project … The downside of this technique is that the learning curve of the game immediately becomes a lot steeper, as the introduction of such process models complicates the simplicity of the game. In the end however, I believe that the increase in educational power would be worth the tradeoff in complexity."

The requirements and design phases of the game received considerable attention in the student essays since the course content focuses on the early stages of the software lifecycle. The common view is that they should be more made interesting and realistic.

- "Another comment on the Problems and Programmers process focus is that it can wrongly uphold the student's view of coding being most important by making most of the action and excitement of the game in the implementation phase. The slow beginning of the game during the requirements and design phases where cards are simply drawn and placed can cause the students to think of these very important parts of the software process as being boring and something that just gets in the way of writing code."
- "Inspections in Problems and Programmers are only addressed in the implementation phase and due to this students can get the wrong idea about the importance of inspections throughout the software process. Although code inspections are a useful way to eliminate defects, along with testing, the inspections of requirements and design could be argued to be much more important, as it is in these phases where mistakes could lead to costly rework to address later."

Several students commented on the competitive aspect of the game. Opinions were divided over whether this enhanced or reduced the game's effectiveness.

- "Rather than having your opponent play problem cards against you, they could be placed in a pile face-down and drawn one at a time during each player's turn. By doing it this way, the problems encountered by the players are more random and it reduces the feeling of your opponent 'sabotaging' your project."
- "The benefit of a competitive game is that players will be able to observe the differences between their own strategies and those of their opponents. The results of these differences will be a powerful example of the consequences of actions taken in a software engineering setting."

REFLECTIONS

I. Impact of the Game on the Course

Introducing the Problems and Programmers game into the software process course was considered a successful innovation. The physical and concrete nature of the game helped illustrate many of the abstract concepts and problems associated with software projects. When these concepts and problems were subsequently discussed in lectures, reference was made to how they were represented in the PnP game.

The many variations within the game demonstrated to students a range of issues that can occur during software development. Using teams to play the game encouraged discussion between students about the concepts and problems and how they can be addressed. The extended use of the game in other activities within the course, such as the software requirements specification assignment, reinforced the initial learning from the game. Students commented that the game helped make them aware of the benefits of documenting and reviewing requirements prior to developing code. Even though only code is explicitly reviewed in the game, the applicability of reviews is broadened by other activities in the course, such as the SRS inspection in the first assignment.

II. Demonstration Game

Because Problems and Programmers is a physical game in which the players must manually enact the game's rules, some overhead in learning and preparing to play the game is unavoidable. Minimizing this overhead should be a goal, as it would facilitate the game's introduction into a course's curriculum. One obvious suggestion for the future is to play a demonstration game in a lecture prior to any tutorial class games. This would ensure that students had some exposure to the game before playing it themselves. A challenge with such a demonstration is to make the cards visible to a large class. A visualizer installed in many lecture theatres at the University of Queensland might make this possible. The players should be familiar with the game so that it is played correctly and without unnecessary delays. It would be useful to have the players adopt different strategies, one playing safely while the other rushes the project by minimizing the requirements and design phases. The learning process would be enhanced if both players were to verbalize the strategy behind their actions as they play the game.

III. Problem Getting Genuine Student Feedback

During the course described in this paper, feedback about the PnP game was primarily collected as part of assessment tasks. An advantage of this approach is that feedback is received from all students, thus giving a complete picture of student perceptions, which is unlikely if feedback is purely voluntary. A disadvantage is that students may feel inhibited or constrained if they perceive that negative feedback could adversely affect their results. In this context, the fact that the game was externally sourced probably reduced this problem.

Courses at the University of Queensland provide students with an opportunity for anonymous feedback at the end of

semester. No negative comments about the PnP game were received, so it seems unlikely that students held negative opinions that they did not feel comfortable expressing in their essays. If they had, the anonymous survey would be the most obvious outlet.

IV. Levels of Realism

The level of realism in a simulation game such as PnP must be chosen carefully. One must ensure that students are not overwhelmed with complex rules, but that they still receive a realistic picture of the process being modeled. The level of realism depicted in the PnP game was chosen carefully, with these considerations in mind. In general, the game strives for realism, but some concessions were made. In some cases, deviations from reality were made to improve game play. For example, the idea that players can use problem cards to hinder one another does not fit into the model of players as competing managers, but this aspect of the game appears to have improved interaction between players and general involvement in the game. In other cases, deviations from reality were necessary to ensure that the rules were understandable. The game's handling of employees' salaries, for example, is greatly simplified.

In general, the PnP game seems to provide an appropriate level of realism. However students' reflective essays indicate that some problems still exist. Some players wished that the game would more accurately depict the software process, while others felt that the game was too complex and difficult to learn. Students generally did not object to unrealistic aspects of the game which added to the game play, but they felt it was a weakness that some aspects of the software process were simply missing in the game's representation. It may be possible to address some of these concerns with improvements to the game's rules.

V. Improving the Game?

As mentioned above, students expressed concerns about the level of realism in the game, some stating that the game was unrealistic, while others felt bogged down in the game's details. These concerns are difficult to address: adding more rules and facets to the game might increase the realism, but could exacerbate problems with learning the game.

This realization revealed limitations to the basic design of the game, which prevented certain aspects of the software process from being easily represented. To explore other possible approaches, a new version of the game is currently being designed which uses a fundamentally different, requirements-based representation of the software process. This has the potential to represent a wider variety of software process models and situations, and continued research will determine if this approach is able to provide greater realism within a more elegant set of rules.

This difficulty also raises another fundamental question about the game: whether it would be more effective at teaching its lessons if implemented as a computerized, rather than physical game. The tradeoffs inherent to this choice are discussed in previous papers [2, 3] and implementation of an electronic version of the revised rule set is planned, which will help to determine the appropriateness of each approach. In the meantime, the second and third authors' work on a separate computerized software process simulation, SimSE [10], will help to shed light on this issue.

ALTERNATIVES

In this section, we briefly review other approaches that are used and advocated for developing student knowledge and skills in the software process.

I. Project Courses

Software project courses are widely used as a method of exposing students to the problems of larger scale software development [1, 4, 8, 13, 14]. Typically, students are organized into teams with a project that extends over a semester, or possibly two. Project courses offer hands-on experience and can involve varying levels of industrial realism, depending on the choices of client, source of requirements, etc. As with the design of simulation games, the challenge is often to balance the realism with the other educational objectives.

The disadvantages of project courses include the one-shot nature of the project with no ability to explore alternative approaches to the same problem. It is quite common to have only a single iteration of the software lifecycle, although there is some anecdotal evidence that this is changing. It is in this regard that simulations, which can be run multiple times in a short period of time, hold an advantage over traditional class projects.

It should be noted that simulations like PnP are not intended to replace software project courses, but to provide a complementary educational tool. As one of the student responses in Section 4.3 above mentions, students need to learn some lessons first-hand. Simulations of this kind can help reinforce the lessons learned via hands-on projects.

II. Watts Humphrey's Personal Software Process

The Personal Software Process [6, 9] is a structured learning experience for software engineers that introduces them to the concepts of software process best practice. The goal is for each student to develop the personal skills and discipline to develop large-scale software products. It uses a progressive sequence of increasingly sophisticated defined processes.

The PSP has the significant educational advantage of requiring active software development though its ten programming assignments that generate personal process data, data that is far more convincing to that individual than data from any other source. This addresses a weakness of some educational simulation approaches such as PnP, which can be less convincing than the type of hands-on experience that PSP can provide. A limitation of the PSP is the focus on personal development so that the issue of scaling up to industrial practice requires extrapolation, at least in the mind of the student.

III. Software Process Simulations

There are several different software process simulators described in the literature, some have research objectives while others have been developed explicitly for education [6, 7, 10, 12]. Invariably these simulators are enthusiastically described by their developers who report success in using them. Collofello [6] describes a simulator using a system dynamics model of the software development process, and how it was used in software project management training. Drappa and Ludewig [7] present the SESAM project, which is probably the most sophisticated software project simulator developed for educational purposes. In their paper, they describe the simulator, their QA model that concentrates on the effects of quality assurance on software projects, and two experiments to see whether using the simulator helps inform project management education. Sharp and Hall [12] report on their interactive multimedia simulation of a software development organization. When using this system, the student "joins" the organization as an employee and undertakes a variety of software engineering tasks.

It is more difficult to find independent reports of successful adoption of software project simulators. Part of the problem may be the failure of the software engineering education community to recognize the value of replication studies. Another difficulty may be the effort to transplant a software process simulator from one educational environment to another.

CONCLUSIONS AND FUTURE WORK

Overall, the PnP card game was a very successful innovation within this software process course. Unlike the previous experiment with PnP, the game was played in teams and was used as part of a larger educational experience, and the game proved to be effective in this role. Problems and Programmers provided the basis for a larger, innovative educational experience. By allowing students to play the game, a solid foundation was established for their SRS documents and reflective essays. The excerpts from these essays indicate that the process as a whole was thought-provoking and effective.

While the PnP game can be readily reused in tutorials for this course in future, the other uses of the game within the course will need some adaptation. The same SRS problem cannot be used in subsequent offerings of the course without raising concern over reuse of previous assignment submissions. Similarly the focus of the individual essay topic must be changed each year to minimize the potential for plagiarism.

Meanwhile, improvement of the game itself continues: a new version of the game is being developed that addresses many of the concerns that students expressed. This new version of PnP focuses on introducing the various software life cycles rather than just a waterfall model, stands to teach more effectively with a different game structure, and could, once again, be used to facilitate SRS exercises and thereby increase student involvement in the course.

ACKNOWLEDGMENT

The authors would like to thank the students in COMP3500 in 2004 for their participation and feedback. Daniel Jarrott, the course tutor, also played a major role in COMP3500 and the use of the PnP card game, and his efforts are greatly appreciated.

REFERENCES

[1] D.J. Bagert. Balancing process and product, Proc. 9th Conference on Software Engineering Education, pp. 78-84, 1996.

[2] A. Baker, E.O. Navarro and A. van der Hoek. An experimental card game for teaching software engineering processes, Journal of Systems and Software,75(1-2):3-16, 2005.

[3] A. Baker, E.O. Navarro, and A. van der Hoek. Problems and programmers: an educational software engineering card game, Proc. 25th International Conference on Software Engineering, pp. 614-619, 2003.

[4] M.B. Blake. A student-enacted simulation approach to software engineering education, IEEE Transactions on Education, 46(1):124-132, 2003.

[5] J. Börstler, D. Carrington, G. W. Hislop, S. Lisack, K. Olson, and L. Williams. Teaching PSP: Challenges and lessons learned, IEEE Software, 19(5):42-47, 2002.

[6] J.S. Collofello. University/industry collaboration in developing a simulation based software project management training course, Proc. 13th Conference on Software Engineering Education & Training, pp. 161-168, 2000.

[7] A. Drappa and J. Ludewig. Simulation in software engineering training, Proc. 22nd International Conference on Software Engineering, pp.199-208, 2000.

[8] D.P. Groth and E.L. Robertson. It's all about process: project-oriented teaching of software engineering, Proc. 14th Conference on Software Engineering Education and Training, pp. 7-17, 2001.

[9] W.S. Humphrey. A Discipline for Software Engineering, Addison-Wesley, 1995.

[10] E.O. Navarro and A. van der Hoek. Design and evaluation of an educational software process simulation environment and associated model, Proc. 18th Conference on Software Engineering Education & Training, pp. 25-32, 2005.

[11] J.M. Rolfe. Flight Simulation (Cambridge Aerospace Series), Cambridge University Press, 1988.

[12] H.C. Sharp and P.A.V. Hall. An interactive multimedia software house simulation for postgraduate software engineers, Journal of Interactive Media in Education, 2001.

[13] J.E. Tomayko. Carnegie Mellon's software development studio: a five year retrospective, Proc. 9th Conference on Software Engineering Education, pp. 119-129, 1996.

[14] V.E. Veraart and S.L. Wright. Software engineering education-adding process to projects: theory, practice and experience, Proc. Asia Pacific Software Engineering Conference, pp. 148-157, 1995.