Incorporating Gaming in Software Engineering Projects: Case of RMU Monopoly

Sushil ACHARYA
School of Engineering, Math and Science, Robert Morris University
Moon Township, Pennsylvania 15108, USA

and

David BURKE
School of Engineering, Math and Science, Robert Morris University
Moon Township, Pennsylvania 15108, USA

ABSTRACT

A major challenge in engineering education is retaining student interest in the engineering discipline. Active student involvement in engineering projects is one way of retaining student interest. Such involvement can only be realized if project inception comes entirely from the student. This paper presents a software game, RMU Monopoly, developed as a project requirement for a software engineering course and describes the challenges and gains of implementing such a project.

The RMU Monopoly was proposed by three junior software engineering students. The game is a multi-platform software program that allows up to eight players and implements the rules of the Monopoly board game. To ensure agility the game was developed using the spiral software development model. The Software Requirements Specification (SRS) document was finalized through an iterative procedure. Standard Unified Modeling Language (UML) diagrams were used for product design. A Risk Mitigation, Monitoring, and Management Plan (RMMM) was developed to ensure proactive risk management. Gantt chart, weekly progress meetings and weekly scrum meetings were used to track project progress. C# and Sub-Version were used in a client-server architecture to develop the software. The project was successful in retaining student interest in the software engineering discipline

Keywords: Software Game, Monopoly, Retention, Education

1. INTRODUCTION

A major challenge in engineering education is retaining student interest in the engineering discipline. This has been a concern for many years. More than 70 years ago student graduation rate stood at 28%, in 1993 the graduation rate stood at 47%, and now the average graduation rate stands at 56% [1]. Like other engineering programs Robert Morris University (RMU) engineering department also has its share of student retention issues. Researchers have mentioned unapproachable condescending faculty [2], inability of schools to admit better students [3], and lack of learning communities [4] as factors affecting engineering student retention rate. At RMU it was felt that having lecture intensive engineering courses did not assist in student understanding, did not provide adequate hands-on real life experience needed for the competitive job market, did not make education interesting enough for students and contributed to reduced student interest in engineering. In view of these issues, as of spring of 2006, RMU's engineering department enhanced all of its engineering courses by incorporating laboratory sessions. Two 50 minutes session per week was allocated for lectures and one 2.5 hours session per week was allocated for lab exercises. This strategic decision was made to ensure that students had adequate hands-on real world experience. After all we tend to retain 70% of what we learn when our involvement is receiving and participating, and 90% when our involvement is being there [5]. Hands-on experience assists in students understanding of processes, methods and tools by mapping theory to practice. In addition all course instructors were given the liberty to incorporate hands-on components like course-based projects, field visits and expert talk sessions into their syllabi.

One such course incorporating all of the listed hands-on components is ENGR3410: Fundamentals of Software Engineering. This is a required junior level course for software engineering majors. However in this course the approach of assigning course-based projects takes into consideration student interest. It is felt that active involvement in course-based projects can only be realized if project inception comes entirely from the student and the student is eager to see project completion. Students are encouraged to propose gaming projects. Introducing games in software engineering is not a new concept but rather one that is being used by many programs to add the fun factor needed to engage students. The growing popularity of computer games coupled with the Computer Science sophistication required to build today's entertainment applications, presents an opportunity to use computer games as a means to better train Software Engineers [6].

This paper presents a software game, namely "RMU Monopoly", developed as a student initiated course-based project requirement for ENGR3410. The paper makes an attempt to present the pains and gains of major activity areas of the Software Development Life Cycle (SDLC) from a student – instructor perspective.

The Need for Course-Based Projects

In order to keep up with the demand for skilled software developers, academia must respond by developing curriculum that fuels the creativity and passion of students. Software Engineering students at RMU are introduced to programming concepts through required courses like C++, Java, and Data Structures. However students are not challenged enough to develop software programs that would further strengthen their understanding of programming methods and tools. One

approach in keeping students motivated is rapid functional development of a software product with the assurance that the product will be publicly displayed. Releasing a program to the public is a major incentive to spend the time required to make quality software products. ENGR3410 recognizes that students need to be challenged to a certain level and uses course-based projects to achieve this. One team released their creativity and passion into building a software version of the Monopoly board game. Figure 1 depicts the game's user interface.

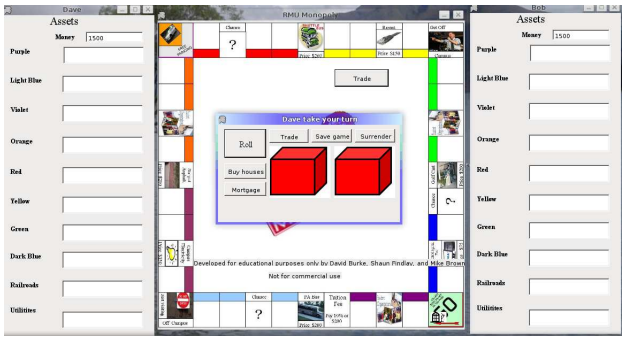

Figure 1: RMU Monopoly

Project Inceptions by Students

Software projects are more interesting to students when the students themselves participate in project inception and decide on what to create. In the case of RMU Monopoly, students choose the project and decided on the implementation methodology. Students were put into teams and asked to propose three possible software projects. The instructor evaluated the proposed projects and selected one project to qualify as a course project.

In section 2 we briefly describe the features of RMU Monopoly. In section 3 we discuss how key software development activities namely, requirement gathering, design, coding, testing and project management were implemented in the context of this project. The challenges and the gains from the student side are reflected. And finally in section 4 we present the project postmortem and conclusions.

2. RMU MONOPOLY

RMU Monopoly is a RMU version of the Monopoly board game developed by three software engineering juniors: David Burke, Mike Brown, and Shaun Findlay. This game was developed as an educational tool and as a game students could play with their friends. The game was designed to make use of pictures and references from the RMU campus. Students playing the game would immediately recognize RMU landmarks and friends (maybe even see themselves in the game). Many game features include inside references that only a RMU student would understand. Furthermore the game is completely platform independent. Students can play on Windows, Mac, and Linux. The game can be edited in any platform. Thus future students will be able to learn from the code, regardless of the platform. The game is played with 2 to 8 players. It features everything one would expect in a regular game of Monopoly. For example properties are bought and traded. Chance cards add some surprises into the mix as well. Here it was decided to vary from traditional Monopoly and make up new chance cards. Often these cards feature a funny story resulting in the loss or gain of money or spaces. All in all, the most important requirement was to have fun in both development and playing. Working on a software project with insider jokes and some degree of silliness is just more fun to program.

3. SOFTWARE DEVELOPMENT ACTIVITIES AND CHALLENGES

A uniqueness in this project is that students were not equipped with all the skills at project start time. Software developed skills were taught in class in parallel to students implementing them on their projects. This meant students were responsible for implementing the skills after they were taught in class.

Software Development Environment

Programming Environment: Students were given the responsibility of deciding the tools to use to develop the game. This gave students the freedom to work in the programming environment of their choice. Though the students had already taken courses on C++ and Java this approach was used to encourage students to try out new tools so as to improve and/or complement their programming skills. However the drawback of this approach was for the students to learn the new tool on their own with very limited support from the professor. Without hesitation, students decided to program in C#. Their decision was based on the fact that C# was platform independent and at least some members of the team had been exposed to C# in the required C++ course. By choosing what to build and how to build it, students took ownership of the project. It was no longer a homework exercise to teach merely a language X, a tool Y, and a concept Z. The project and tools for creating it belonged to the students. However a major challenge the students faced was in learning C# to be able to program games. Students acquired two C# books and relied heavily on internet resources. Students also taught each other anything they knew about the language that they could use to meet their objectives. C# was not the only challenge however. Learning general programming techniques was a considerably more time consuming task. Difficulty in learning to work with graphical user interfaces, threading, and communication between classes were all noted in the post mortem report as being very time consuming.

Software Design Studio: Many of RMU's computer labs are restricted as a defense against students installing inappropriate software. However the Software Design Studio (SDS) did not have such restrictions. The software design studio is setup to serve the student body in a unique way. Software engineering students are authorized to install and uninstall software for education purposes. Students wanted to run a Subversion server, and were freely able to do so. Students preferred OpenOffice.Org to the Microsoft Office already installed on all school computers, so the students were able to install it themselves. Upon request, Visual Studio 2005 was installed to the computers being used. None of this would have been possible had the school setup strict guidelines on computer usage. The students were even given after hours access into the SDS. The freedom to use the computers in the way students wanted to, assisted immensely in the success of the project.

Hardware Environment: Hardware requirements were well defined. Students wanted the game to run on all

major platforms, Windows, Linux, and Mac, and so they needed the platform specific machines to test their program. Unfortunately only Windows and Linux machines were available in the SDS. A hardware requirement was that it must be able to run .NET or Mono. However as a Macintosh machine was not available in the SDS this could not be tested. Another hurdle was that the school did not allow the SDS's Linux server to be accessed off campus for security reasons. Since the students used this server for hosting their subversion repository, this handicapped students ability to work remotely. This issue was resolved by doing most of the work on campus .

Research and Requirements Analysis

After the inception the first software development activity carried out was research and requirements analysis. The output of this activity was a Software Requirement Specification (SRS) document. The students spent time researching the game of Monopoly. The students had played the game before and knew the basic rules of play. Still the research provided each student a better understanding of the game. For the requirements analysis activity the students were asked to play a dual role of a customer and a software developer so as to effectively define the requirements and the project scope. Requirements engineering was taught in the lecture part of class and the students used this theoretical understanding for requirements analysis in a lab session. The students performed elicitation, analysis, specification and validation of the requirements. This forced students to really think hard about their project. It changed the ambiguous project of RMU Monopoly into a well defined project with adequate features. Students choose to include features like computer controlled players and real photos taken around campus to use in parts of the game. Students also surveyed their friends in what they wanted in the game. To make the project have real life flavor, the professor implemented "creeping requirements" by adding a new requirement as the students were beginning to work on the design phase. The new requirement was to include video streams in the game. This was added to simulate the changing requirements of real world customers. All of this stimulated students to really think about and get involved with the project. Students also decided on the scope of the project.\

Software Design

It was decided that the spiral software development model would best fit a project of this nature. The spiral model allows agility and easy removal of requirements when running behind schedule. The model also helped students make better estimations. Unified Modeling Language (UML) diagrams were used for software design. Students created a use case (Figure 2) and class diagrams (Figure 3) for the project.

Software Coding

Coding the program took a significant amount of time. Code was divided up into modules. At weekly scrum meetings the team discussed which modules needed to be done and which were ready. The team assigned who would be in charge of each module and design them too. Most of the design work was ad hoc, written on white boards. Coding the project required large amount of time just in research. The team decided to try something like threading, when no one had actually used threading before. This made coding the hardest part of the project. The only way to make up for this deficiency was to spend more time coding and researching how to code.

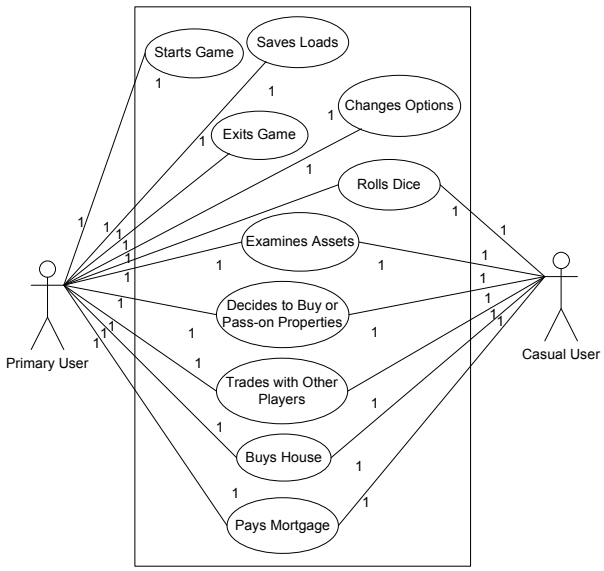

Figure 2: A Use Case diagram

The students were used to writing programs with only a few hundred lines of code, but RMU Monopoly took thousands of lines of codes (KLOCs). Students used Sub-Version to control the source code and distribute it. They also used object oriented design to ease in the design. Modules that required many replications of data, such as data about each player, were made much easier with classes and objects.

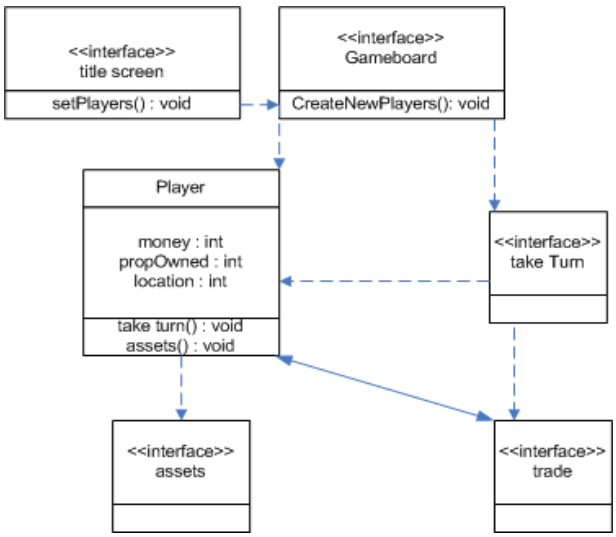

Figure 3: Early draft of a Class diagram

Software Testing

During the duration of this project informal unit and integration tests were carried out. However no formal testing took place. This was due to the fact that students had no knowledge of formal testing and the deadline for delivery did not allow for testing. The students did however use the same project for a Validation and Verification class taken the following semester. In this class unit and integration testing were taught in detail and practiced on the RMU Monopoly program. Some students decided for themselves to continue working on fixing the many bugs found, despite no credit being offered. Students decided

for themselves to host the code to Google Code and start an issues tracking system on it to monitor the bugs found. NUnit was used to create unit test cases for the program. Also some manual testing was performed to find other issues. Students took part in an inspection meeting for one lab in the class. During the inspection a RMU alumni came in to talk about how his company did inspection meetings. While ideally this testing should have been done concurrently with development, time and lack of adequate knowledge forced this to be placed in a separate class.

Project Management

The project was implemented mimicking a true software development environment. Students took on self selecting roles in the project, including requirements manager, design manager, and code manager. These roles were not enforced and as needs arose the students took additional roles that suited them best. Some new roles were graphic artist and project manager. The professor became the customer and the students gave bi-weekly presentation on how the project was progressing. The presentation came complete with prototypes of the game, which was easy to do with the spiral development model. Besides the professor another category of customers were the students' colleagues in the same class and the members of the RMU student chapter of the Association of Computing Machinery (ACM). These colleagues looked at the game and provided constructive suggestions. Many students outside the development group also became partially involved when the developers took photos of them to be used in the game. This technique created a link between the developers and the "customers" which strengthened the game design. It also provided a real life experience to show the importance of good communication with the customer.

Project Estimation: Estimations can be very challenging to students who have no real world experience to back up estimations. However as mentioned earlier the Spiral model assisted students in making project estimations. The goal was to keep a 40-20-40 time distribution for the three development activities: design, coding, and testing. However with the challenges in learning new concepts of a programming language the time distribution had to be regularly re-estimated. At project completion requirements gathering, research and design required 70% of the total time. Research involved both understanding the Monopoly game as well as learning new programming concepts. Likewise coding and testing required 20% and 10% respectively of the total time. Figure 4 depicts the final time distribution. In this chart the research component involves understanding Monopoly as well as learning programming concepts.

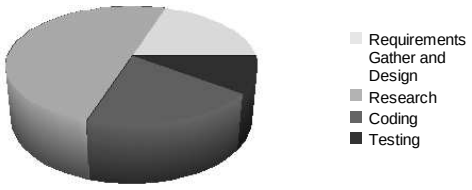

Figure 4: Time allocation

Project Schedule: A project schedule (Gantt chart) was created to ensure that tasks and subtasks were properly understood and resources were adequately assigned. Like any other project, scheduling was done to keep track of the project. However changes in project scope required reworking of the schedule towards the delivery deadline. Figure 5 depicts a portion of the Gantt chart created for this project.

RMMM Plan: A Risk Mitigation, Monitoring, and Management plan (RMMM) was developed to ensure proactive risk management. This included what requirements could be scrapped or down scaled if the project went behind schedule. The project schedule and weekly progress meetings were used to keep track of project progress. On days when work was done (mostly weekends and late at night) a scrum meeting was held to review progress and set goals for the day. When the project did fall behind schedule, students were immediately aware and made informed decisions on how to get back on track. However the initial RMMM plan had to be changed as new challenges became visible.

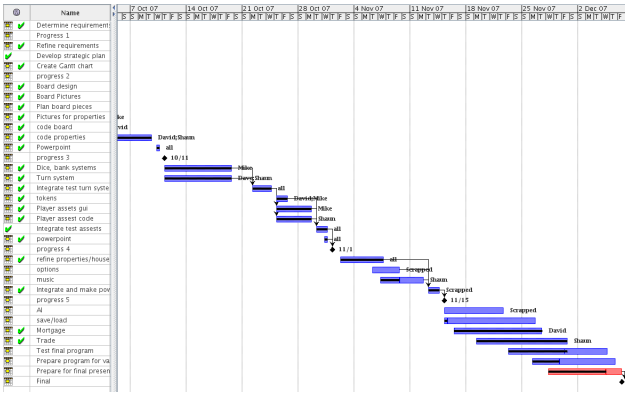

Figure 5: RMU Monopoly Gantt chart

4. PROJECT POSTMORTEM AND CONCLUSIONS

Project ownership namely a software gaming project was the key factor in retaining student interest in the software engineering process. This ownership created more of a drive to finish the project than would a professor initiated course-based project. The best example of this commitment was shown during the latter part of development. Near the due date of the project, it was decided that certain features had to be dropped and/or scaled down in order to make a working game. These dropped features wouldn't necessarily mean a bad grade, since the students were also being graded on participation and the understanding of the software engineering concepts being used. However it was decided that, despite there being not enough time, the feature to “trade properties with other players” was crucial for an enjoyable game of monopoly. Another hurdle that could stop an uninterested student was learning a number of new tools for the project. C#, AgroUML, Mono (a .NET implementation for Linux and Mac), and Sub-Version were all new tools for the students. However, with interest in making the best gaming program possible, students shrugged off the necessary learning curve of these tools. Another factor in handling these new tools was that by using the spiral model, prototypes were made, inspiring the students that they can use such tools to make real results. The problem could be mitigated more by teaching a variety of tools and by having a knowledgeable pool for guidance, so that answers to student questions could be easily available.

There are unfortunately a number of challenges to implementing a project similar to RMU Monopoly. A genuine interest in software is needed for students to take interest and ownership of their work. The best type of student for this type of project is one that would probably be programming even if they were not in school or work. While incorporating gaming can help students gain interest in software development, it is up to the student to commit. Also implementing this type of project in a larger class size may be challenging. The RMU Monopoly project was done in a small class size. This allowed for individual attention from the professor. In a larger class, it may be more tempting to assign one generic project that every student must complete. While this would make grading and teaching easier, it would strip the students of ownership and interest of their project.

In the experiences at RMU, retaining student interest in software engineering is vital to successful learning. Course based gaming software projects like RMU Monopoly a gaming software was a successful means of keeping student interest in the SDLC. By letting students choose what their project will be and how to implement it, ownership of the project was given to the students. Overall this resulted in a functional game and a superior learning experience with a fun factor for the students. This method could easily be adapted to suit other colleges in the effort to attract, educate and retain future software engineers.

5. REFERENCES

[1] Knight, D.W., etc. al., "Improving engineering Student Retention through Hands-On, Team Based, First-Year Design Projects", **Proceedings 31st International Conference on Research in Engineering Education**, ASEE, June 22-24, 2007, Honolulu, HI.

[2] Vogt, C. M., "Professors Need to Lighten Up", **ASEE PRISM**, March 2008, Volume 17, Number 7.

[3] Huband, F.L., "Attracting best – and Keeping Them - Comments from the publisher", **ASEE PRISM**, February 2008, Volume 17, Number 8.

[4] Meyer, J., et. al., "Retaining Freshman engineering Students through participation in a first-Year Learning Community: What works and what doesn't", **Proceedings of the 2007 American Society for Engineering Education Annual Conference & Exposition,** Copyright 2007, American Society for Engineering Education

[5] Morse, L.C. and Babcock, D.L. **Managing Engineering and Technology**, 4th Edition, Prentice Hall International Series in Industrial and Systems Engineering, Editors - Fabrycky, W.J. and Mize, J.H., 2007.

[6] Claypool K., and Claypool M., "Teaching Software Engineering through Game Design", **Proceedings of the 2005 conference on Innovation and Technology in Computer Science Education (ITiCSE)**, 2005 June 27-29, Monte De Caparica, Portugal.

[7] Welch, L. R., etc. al., "Enhancing Engineering Education with Writing-to-learn and Cooperative Learning: Experiences from a Software Engineering Course", **Proceedings of the 2002 American Society for Engineering Education Annual Conference & Exposition,** Copyright 2002, American Society for Engineering Education.

[8] Wankat, P. and Oreovicz, F., "Making them want to stay", **ASEE PRISM** Volume 14, Number 7, 2005.