Using Student Incepted Projects to Retain Student Interest in Software Engineering

by

Sushil ACHARYA
acharya@rmu.edu
Department of Engineering
Robert Morris University

Shaun FINDLAY
svfst2@mail.rmu.edu
Department of Engineering
Robert Morris University

David BURKE
dmbst32@mail.rmu.edu
Department of Engineering
Robert Morris University

Mike BROWN
mabst45@mail.rmu.edu
Department of Engineering
Robert Morris University

Abstract

The courses we teach and the pedagogy we use are crucial in retaining student interest in an engineering program. A noble approach to retaining student interest in software engineering through student initiated course-projects is described. A pedagogy that qualifies student incepted ideas as course-based projects, coaches' students to use industry best practices, teaches students just-in-time skills, provides students a certain degree of development freedom and makes learning fun is discussed in the context of developing a software game "RMU-Monopoly". The pains and gains from an instructor-student perspective in the software development process are presented in detail. Four years of pedagogy implementation suggests the merit of our approach, showing increased student in the program and boosting retention rates.

Keywords: Retention, Software Engineering, Games, Monopoly

I. Introduction

Student retention has been a major concern not only for the many U.S. engineering programs but also for the engineering workforce. Only 56% of undergraduate students admitted nationally are retained to graduation, with some schools reporting retention rates of only 30% [1]. Clearly, engineering education continues to face a significant challenge in retaining qualified students, especially those who bring creativity and innovative thinking to their studies but are disaffected by their pedagogical experiences in the early stages of the engineering curriculum [2]. Low rates of student retention within the discipline have heightened concerns in the engineering community about the structure, content, and engineering education [1]. Without refocusing and reshaping the undergraduate engineering learning experience, America's engineering preeminence could be lost [by 2020] [3]. With the dependence the US economy has on innovation, a talent shortage will have broad impact on what is becoming an increasingly competitive global market [4].

Authors, theorists and researchers have found six themes: *i. boredom and uncertainty, ii. transition/adjustment problems, iii. limited and unrealistic expectations of college, iv. academic under preparedness, v. incompatibility* and *vi. irrelevancy* to be the reasons why students do not return after their freshman year [5]. Some researchers have mentioned unapproachable condescending faculty [6], inability of schools to admit better students [7], and lack of learning communities [8] as factors affecting engineering student retention rate. As engineering curriculum is very demanding students may leave due to poor academic performance or conclude that the heavy demand of the curriculum is not worth a continued effort [9].

Some institutions have initiated innovative programs that are boosting retention rates. The College of Engineering at Southern Illinois University, Carbondale has initiated a program through a $1.5 million grant from the National Science Foundation that is expected to improve the overall graduation rate in the college to 67% from the existing 38% [10]. One of the key activities planned by the institute is requiring freshmen and sophomores to live on campus in one of three designated residence halls. The Neumont University in Salt Lake City has won a CIO 100 award for their innovative approach to educating industry-ready software developers and engineers [11]. The well received "project approach" the university takes requires 70% involvement in real-world projects. Neumont plans to scale the salt lake campus to around 2,000 students and graduate about 40% of the student body per year which in the software field is a high number. At Robert Morris University (RMU) course-based projects and team-based learning have been introduced as essential components in the laboratory sessions initiated by the university to retain student interest in engineering and to prepare "industry-ready" graduates.

This paper presents a noble approach to retaining student interest in software engineering through student initiated course-based projects. Fewer students leave engineering studies when education programs link concepts to real-world practice [1]. Course-based projects provide student just-in-time skills to be creative problem solvers and life long learners. In addition the project instills upon students the value of teamwork, communication and project management. 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. The case study presented in this paper is an implementation of a software version of the Monopoly board game, namely "RMU Monopoly". This project is a requirement for, ENGR3410: Fundamental of Software Engineering, a software engineering course at Robert Morris University (RMU). 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 [12].

In the following sections this paper describes the pedagogy used and presents the pains and gains of major activity areas of the software development process from an instructor-student perspective. Section 2 describes the pedagogy behind this work. In section 3 the features of RMU Monopoly is explained. In section 4 key software development activities namely, requirements gathering, design, coding, testing and project management implemented in the context of this project are described. Also in section 4 the challenges encountered and the pains

and gains from the student side are reflected. And finally in section 5 we present the project assessment and recommendations are presented.

II. The Pedagogy

In order to boost retention rates and to keep up with the demand for skilled software engineers, academia must respond by developing curriculum that fuels the creativity and passion of students. The courses we teach and the pedagogy we use are crucial in retaining student interest in the engineering program. Universities are promoting student engagement as a means to increase retention [13]. The pedagogy used in ENGER3410 is as follows:

- *Form project teams:* Class is requested to form teams consisting of 3 or 4 students. Strengths and weaknesses of teams are evaluated (by the instructor) and if required the teams are reconfigured through mutual understanding.
- *Request for project ideas:* Project teams are given three days to discuss and propose three possible course-project ideas. They are told that the project needs to meet specific software engineering requirements, has to be completed in 13 weeks and will be displayed in the "Engineering Wall of Projects" when successfully completed. Students are also told that the teams will own their projects.
- *Evaluate and qualify project ideas*: The project ideas are evaluated on a set of criteria: Will there be adequate activities, tasks and subtasks? Can a project schedule be defined? Will the students be able to gather enough requirements and define a scope? Can a Risk Mitigation, Monitoring and Management (RMMM) plan be created? Can design diagrams (Class, Sequence, Use case) and User Interfaces be created? Can data tables be defined? Will the project have enough resources? Do the team members have minimal skills?
- *Teach "Just-in-time" skills:* As the students implement their projects during lecture sessions they are taught fundamental software engineering skills. In addition laboratory sessions are used to develop work products like: Software Requirements Specification (SRS), Software Design Document (SDD), RMMM and project schedules. Students are also expected to implement the project using software engineering best practices.
- *Introduce real world activities:* Students are encouraged to conduct different types of meetings (work meetings, scrum meetings, inspection meetings) and provide regular progress updates (formal presentations, email updates, ad hoc updates). They are also introduced to creeping requirements, team conflicts and altered deadlines.
- *Facilitate project execution:* Students need to have the tools to perform. Students are provided unlimited access to laboratory room and to the development tools.
- *Prepare project poster:* A project poster comprising of the various software artifacts is prepared and displayed in the engineering department.

The key to this pedagogy is "student incepted course-based projects". In a traditional classroom setup the educators unilaterally make the final decision on the content and direction of a course-based project. However software projects are more interesting to students when the students themselves participate in project inception.

Student incepted projects alone do not guarantee interest and motivation. An approach to keeping students interested and motivated is rapid functional development of their software product with the assurance that the product will be publicly displayed. In addition to a course grade, releasing a work product to the public provides a major incentive to spend the time required to make quality software products.

During the course of project implementation students are given the freedom to make decisions and to learn from mistakes. The approach focuses on problem-based learning and extensive mentoring practices. The educator is not only an instructor but also a coach. Students learn to think instead of learning only by rote. As the project advances students learn to implement "Just-in-time" skills.

III. RMU Monopoly

A team of three software engineering juniors released their creativity and passion into building a RMU version of the Monopoly board game, namely "RMU Monopoly". The game is completely platform independent and can be played and edited on Windows and Linux. This enables future students to learn from the code, regardless of the platform. The game features include references to RMU. Students studying the game would immediately recognize RMU landmarks and friends (maybe even see themselves in the game). The game incorporates all standard features of the regular board game like buying and trading properties as well as chance cards. However new 'chance cards' are developed to implement funny stories resulting in the loss or gain of money or of spaces. The game can be played with 2 to 8 players. In the words of students *"working on a software project with insider jokes and some degree of silliness is just more fun to program"* and *"the most important requirement was to have fun in both development and in playing"*.

IV. Software Development Activities and Challenges

Integrating projects into courses is challenging however it benefits both the students and the faculty. The challenges that arise are a reflection of real world scenarios. Uniqueness in this course-based project is that students are not equipped with all the skills at project start time. Software development skills are taught in class in parallel to students working on their projects. This means students are responsible for implementing acquired skills after they are taught these skills in class. In the following sections we discuss the different software development activities carried out in implementing this project.

a. Software Development Environment

- **Programming Environment:** The approach used in this pedagogy is to give the students the responsibility to decide on the programming environment. This provides the students the freedom to work in an environment of their choice. In the case of RMU Monopoly, as the students had already taken courses on C++ and Java they were encouraged to try out new tools so as to improve their programming skills. However the challenge of this approach was for the students to learn the new tool of their choice on their own with limited support from the professor.

  Without hesitation the students decided to program in C#. Their decision was based on the fact that C# was platform independent and one member of the team had been exposed

to C# when he studied C++. By choosing what to build and how to build it, the 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. Students acquired C# books and relied heavily on internet resources to learn C# to be able to program games. Students also taught each other anything they knew about the language that they could use to meet their objectives. However learning C# was not the only challenge. Difficulty in learning to work with graphical user interfaces, threading, and communication between classes were all noted in a 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) does 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 following certain guidelines. For instance students can only uninstall programs that the student installed and no one else uses.

  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 the lab computers, the students were able to install it themselves. Upon request, Visual Studio 2005 was also installed. None of this would have been possible had the program setup strict guidelines on computer usage. The students were even given unlimited after hour access to the SDS. In the words of students *"the freedom to use the SDS in the way we wanted to assisted immensely in the success of the project"*.

- **Hardware Environment:** Students are required to prepare a draft list of hardware requirements at the beginning stage of the project and make modifications if necessary as the project advances.

  Students wanted the game to run on all major platforms (Windows, Linux, and Mac), and needed platform specific machines to test their program. Unfortunately as a Mac machine was not available in the SDS the students were not able to run Mono (a .NET implementation for Linux and Mac). 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 student's ability to work remotely. However the students resolved this issue by doing most of the work on campus.

b. Research and Requirements Analysis

After project inception the first software development activity carried out is research and requirements analysis. The output of this activity is a SRS document. For the requirements analysis activity the students are 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 gathering is taught in the lecture part of class and the students use this theoretical understanding for requirements analysis in a lab session. Creeping requirements are also introduced to emulate the real world customer behaviors.

The students spent time researching the game of Monopoly. Even though the students had played the game before and knew the basic rules of play, research provided each member a better

understanding of the game. The students performed elicitation, analysis, specification and validation of the requirements. This forced students to really think out of the box. The exercise 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 the campus. Students also surveyed their friends in what they would want in the game. To emulate a real life flavor, "creeping requirements" were introduced a week after the initial set of requirements were finalized and the students had started work on the design phase. The requirement added was that the game shall incorporate at least five video streams of RMU landmarks. In the words of students *"all of this stimulated us to really think about and get involved with the project"*.

c. Software Design

Software process models and their use, advantages and disadvantages are discussed in class. Students are asked to decide on the model they will use based on the product they are building and justify their selection. Students are also taught standard Unified Modeling Language (UML) and are asked to use UML standards for their product design.

The students decided that the spiral software development model would best fit a project of this nature as it allows agility, scope management and easy requirements downsizing if necessary. This model also helps students make better estimations. Students created use case (Figure 1) and class diagrams (Figure 2) for the project. In the words of the students *"the UML diagrams were extremely useful in converting user requirements into design documents"*.

Figure 1: A use case diagram

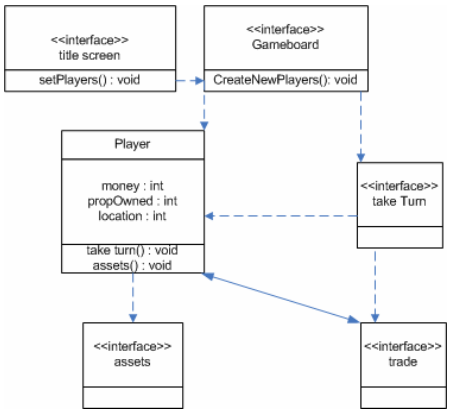

Figure 2: Early draft of a class diagram

d. Software Coding

In this pedagogy the students decide on a programming environment. The professor helps in the selection of the environment but provides only minimal support. Students are coached to use a modular approach, divide work and discuss often.

In order to write effective and efficient codes the students spend considerable time researching concepts. The team decided to try concepts like threading, when no one had actually used threading before. This made coding the hardest part of the project. In the words of the students *"the only way to make up for this inefficiency was to spend more time coding and researching how to code"*. To simplify the process, modular development approach was used. Features were

divided into modules and assigned to team members. The modules were owned by the assigned teammate who was responsible for seeing its completion. The development statuses of the modules were briefly discussed in frequently held scrum meetings and in detail in the design inspection meetings. Besides the UML diagrams additional design work was ad hoc and written on white boards. The students were used to writing programs with only a few hundred lines of code (LOC), but RMU Monopoly took thousands of KLOC. Certain tools and methods helped the students as they progressed. Students used Sub-Version to distribute and control the source codes. Students also used Object Oriented Design to simplify the design. Modules that required many replications of data, such as data about each player, were much easier to implement with classes and objects.

e. Software Testing

This is the first course the students take in a series of software engineering courses. Software testing is not covered in detail in this class as it is covered in an "ENGR3400: Software Verification and Validation" course. However in this class students are given an overview of unit testing and integration testing.

During the duration of this project informal unit and integration tests were carried out. Formal testing did not take place since the students did not have the necessary skills. In the words of the students *"incorporating some testing as part of development assisted us in identifying errors way earlier"*.

f. Project management

The students are taught the importance of project, process, people and product for a successful software project. Project estimations, project scheduling, risk management and change management are also discussed in detail in class.

The project was implemented mimicking a true software development environment. Students took on self selecting roles in the project, such as requirements manager, design manager, and code manager. These roles were not enforced and as needs arose the students took additional roles that best suited their skills. Some new roles were graphic artist and project manager. The professor became the customer and the students gave bi-weekly presentation on the project. 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 student colleagues in the same class and the student members in the RMU Association of Computing Machinery (ACM) student chapter. These colleagues provided constructive suggestions as well. 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 on the importance of good communication with the customer. Project management activities implemented are as follows:

- **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 development model assisted students in making project estimations. The goal was to keep a 40-20-40 time distribution respectively for the 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.

- **Project Schedule:** Project scheduling was done to keep track of the project. A Gantt chart was created to ensure that tasks and subtasks were properly defined and resources adequately assigned. Weekly meetings were used to keep track of project progress. On days when work was done (mostly weekends and late at night) meetings were held to review progress and set goals for the day. When the project fell behind schedule, students were immediately aware and made informed decisions to catch up. The changes in project scope required reworking of the schedule towards the delivery deadline

- **RMMM Plan:** A Risk Mitigation, Monitoring, and Management (RMMM) plan was developed to ensure proactive risk management. Risk description, category, probability, impact and mitigation plans were developed. The plan included provisions for requirements downsizing or dropping. As new challenges like time constraints, hardware issues and programming concepts became obvious the initial RMMM plan had to be modified.

V. Project Assessment and Observations

Project ownership was a 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 more typical course-based project. The best example of this commitment was shown during the latter part of development. Near the project deadline when time became a constraint the students decided that certain features had to be dropped and/or downsized in order to make a working game. These dropped features wouldn't necessarily have a major impact on the grade as the students were also being graded on participation, decision making abilities and the understanding of the software engineering concepts being used. In addition to dropping certain features students also reprioritized the requirements and decided that certain features would have to be developed at any cost. For instance the feature to "trade properties with other players" was crucial for an enjoyable game of Monopoly. Students worked long weekends even into the night in order to complete this feature. In the words of the students *"it was the motivation to make a quality game that drove us to go beyond what was required of us"*.

Another hurdle that could stop an unmotivated student was learning a number of new tools for the project. C#, AgroUML, Mono, and Sub-Version were all new tools for the students. However, with motivation in making the best program possible, students shrugged off the necessary learning curve of these tools. By using the spiral development model, prototypes were made, inspiring the students that they can use such tools to make real results. In the words of the students *"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". Overall this student incepted project resulted in a functional game and a superior learning experience.

This method could easily be adapted to suit other colleges in an effort to attract, educate and retain future software engineers. There are a number of challenges to implementing a project similar to RMU Monopoly. A passion in software is needed for students to take genuine 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 class size greater than 15 may be challenging. The RMU Monopoly project was done in a class size of 10. 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 of the project and dilute their interest in software engineering.

VI. Conclusion

In the experiences at RMU, student incepted projects have been successful in keeping student interest in the software development process and in the software engineering program. The pedagogy has been well received by the students. In fact after the second implementation cycle of this pedagogy, students come to class prepared to form a team and to propose their project ideas. School Scheduling System, Fuel Perk System, Point of Sales System, Insurance Management System, Inventory Control System and A Chemistry Tool Kit are some of the projects students have proposed and successfully completed.

Table 1: Transfer Students by Academic Year

Academic Year	Transfer %
2005-2006	30%
2006-2007	0%
2007-2008	40%
2008-2009	37%

In the years prior to the implementation of this pedagogy student dropout rates at the junior level stood at approximately 5%. However with the implementation of this pedagogy 100% of the students have been retained and student enrollments in the sophomore and junior levels have increased through transfers (refer to Table 1). During this period only one student has dropped out due non-academic reasons. Students have cited their desire to work on software engineering projects of their choice as one of the reasons for transferring. Four years of pedagogy implementation suggests the merit of our approach, showing increased student in the program and boosting retention rates.

References

[1] Fortenberry, N. L. et. al., "Engineering Education Research Aids Instruction", ***Science***, 31 August 2007, Vol. 317. no. 5842, pp. 1175 – 1176, DOI: 10.1126/science.

[2] Bernold, L.E., et. al., "Understanding Our Students: A Longitudinal Study of Success and Failure in Engineering with Implications for Increased Retention", ***Journal of Engineering Education***, 2007, ASEE, pp 263-274.

[3] "National Academy of Engineering, Educating the Engineer of 2020: Adapting Engineering Education to the New Century", ***The National Academy Press***, 2005.

[4] Stephenson, et. al., "Increasing Student Retention in Computer Science through Research Programs for undergraduates", ***SIGCSE'07***, March 7-10, 2007, Covington, Kentucky, USA, Copyright 2007 ACM.

[5] Johnson, E.B., "A study of retention of freshmen students in the college of engineering and applied science at the university of Wisconsin-Milwaukee", ***UWM –CEAS Publication***, 2005.

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

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

[8] Meyer, J. et. al., "Retaining Freshman engineering Students through participation in a first-Year Learning Community: What works and what doesn't", ***American Society for Engineering Education***, 2007.

[9] Anderson-Rowland, M., "Understanding Freshman Engineering Student Retention through a Survey", ***ASEE Annual Conference***, 1997.

[10] Crosby, T., "Engineering's Retention Effort Shows Success", ***The Saluki Times***, October 29, 2008, Southern Illinois University Carbondale.

[11] McKinley, S., "Project- Based Learning: The Neumont University Story, Ubiquity Information Everywhere", ***ACM-Ubiquity***, 2005, Volume 6, Issue 41, http://www.acm.org/ubiquity.

[12] Claypool K., and Claypool M., "Teaching Software Engineering through Game Design", ***ITiCSE '05***, 2005 June 27-29, Monte De Caparica, Portugal.

[13] "Promoting Engagement for all Students: The Imperative to Look Within", ***2008 Results - National Survey of Student Engagement***, Indiana University Center for Postsecondary Research, 2008.

[14] Acharya, S. and Burke, D., "Incorporating gaming in software engineering projects: Case of RMU Monopoly", ***The 2nd International Multi-Conference on Society, Cybernetics and Informatics***, June 29th – July 2nd, 2008, Orlando, Florida, USA.

[15] Wankat, P. and Oreovicz, F., "Making them want to stay", ***ASEE PRISM***, 2007 Volume 14, Number 7.