AC 2011-705: MULTIPLAYER ON-LINE ROLE PLAYING GAME STYLE GRADING IN A PROJECT BASED SOFTWARE ENGINEERING TECHNOLOGY CAPSTONE COURSE

James N. Long, Oregon Institute of Technology

James Long is an associate professor in software engineering technology. Courses and interest are Software Engineering Project Course, Computer Networks, Operating Systems, Embedded Systems and applications. James is the program director for the Embedded Systems Engineering Technology program.

Linda Sue Young, Oregon Institute of Technology

Professor Linda S. Young has taught at the Oregon Institute of Technology since 1983. She earned her Ph.D. in Rhetoric and Composition from the University of Washington in 1997, and has a master's degree from the University of Virginia. She has served as department chair of the Communication Department and has taught a wide range of writing and communication courses. She was instrumental in the design of the bachelor's degree in Communication Studies at OIT. Areas of interest include the overlap of game design and learning systems, media and the communication styles of Japan, creativity and communication, and conversation analysis.

Multiplayer On-Line Role Playing Game Style Grading in a Project Based Software Engineering Technology Capstone Sequence

Abstract

Software systems analysis, design, and construction are tacit activities. As any large software system is developed, the engineers involved in the development activity learn as they go about what it is they are building. Previous experience helps in getting the job done; however, technology, tools, and end user desires change so rapidly that a project may transform several times during the course of system development. Students who gravitate toward the field of software engineering seem to be drawn to it because they enjoy completing ambiguous tasks requiring knowledge gained through the engineering experience. These students are also often drawn to multiplayer on-line role playing games (MMORPG). The nature of the knowledge workforce is changing based on "millenials" entering college and their habits related to playing Internet based games[1]. According to a Pew Research study, game playing is ubiquitous among American teenagers. Fully 99% of boys and 94% of girls between the ages of 12 and 17 report playing video games.[2] This population is entering universities bringing Internet-based learning styles and experience into a lecture-based learning environment. Standard lecture style teaching methods do not match the tacit work environment required of software engineering professionals. This paper explores the application of MMORPG scoring techniques to course topic introduction, curriculum flow, and grading for a year long, project based, software engineering technology capstone course offered in the junior year. Students are formed into teams of three or four; then they are set free to discover information about the "tasks." These "tasks," if completed correctly, will gain them the ultimate position of "Lead Software Engineering Architect". Students are involved in a yearlong odyssey targeted at large scale software project management and self discovery of techniques required to build a successful system. The paper discusses incoming student demographics, course structure, use of knowledge gold and experience points as incentives, project approach, and outcome of this curriculum management model. A method for assessing student learning is discussed along with results. Student attitudes and thoughts are also explored.

Program Overview

The Computer Systems Engineering Technology department of Oregon Institute of Technology offers a four year Bachelor of Science degree in Software Engineering Technology. Oregon Institute of Technology is accredited through the Northwest Association of Schools and Colleges (NWASC). The Software Engineering Technology degree is accredited through the Accrediting Board for Engineering and Technology (ABET).

The Software Engineering Technology program currently has approximately 100 students enrolled in the four years. The program draws students directly out of high school. Students normally have exposure to computer programming and a high level of interest in becoming "programmers". Students who gravitate towared the technical program offered at Oregon Institute of Technology over computer science degrees offered at larger state universities are interested in hands-on activities in a highly interactive environment. By the end of the second

year, students are well versed in programming skills, so there is a need to redirect "programming" ideas toward interest in becoming "software engineers". The third year of the Software Engineering Technology program focuses on systems analysis, design, construction, deployment, testing, and quality assurance. The core of this activity is a three term course sequence encompassing team based construction of a real-world enterprise scale system. The projects are based on problems derived from on campus research projects or industry partners.

Course Modification Motivation

Over the past eight years, student motivation and satisfaction with the three term, project based, software engineering course has steadily declined. Low student motivation was making it more difficult to achieve key learning goals. According to recent assessments by the Junior Project sequence instructor, the course material was no longer effective to the extent it had been when first introduced. Student behavior changed, and the experience level of those students entering their junior year also changed. These factors prompted the following questions:

- What do these students do well as they enter their junior year?
- What activity will engage the current student population in light of the weakening effect of standard lecture, example, and exercise?

In an attempt to answer these questions, an informal questionnaire was passed to the junior class finishing the course sequence in June, 2010. The one component consistent with the student population was experience with on-line role playing games. Jane McGonigal has cited these statistics (TED talk, Feb 2010):

- 3 billion hours a week are spent by people playing online games.
- As of Spring, 2010, the U. S. population has spent 5.93 million man years playing *World of Warcraft*.
- By age 21, the average person will have spent 10,000 hours playing on-line games.
- For the average child in the United States, 10,080 hours is the cumulative amount of time spent in school from 5th grade till high school graduation assuming perfect attendance.
- According to Malcom Gladwell's book, *Outlyer*, any individual spending 10,000 hours at an endeavor by age 21 will be a virtuoso at that endeavor.[3]

The current Internet enabled technical environment is raising a generation of Virtuoso gamers. How can this talent be captured to obtain a useful outcome? In universities, it is widely accepted that much learning occurs outside the classroom, but universities have no coherent gaming the system strategy for leveraging that edge activity[4].

To gain some insight on the current software engineering technology junior class and how applicable the above statistics are to the population under consideration, a survey was given. The results were:

- 95%, of the students either currently spend time playing MMORPG or have had experience in the past playing MMORPG.
- 85% of those students surveyed currently enjoy these activities and will spend a large amount of their leisure time engaged in playing MMORPG.
- 95% of the students said they learn best from a high level of trial and error rather than reading, exercises, and application of theory.

With this information in hand, a re-structuring of the Junior Project course sequence was started. The idea was to provide the standard software engineering curriculum with a MMORPG gaming twist. The software engineering activities did not involve virtual worlds, and support for such an environment was out of reach of the researchers. The scoring mechanism was, however, relatively easy to emulate and adaptable to standard percentage scale grading. Adaptation was done in a sequencing of information introduction and designing point accumulation where discovery, trial and error, and continuous feedback on task outcome were the goals. Each step along the way involves the following components of MMORPG

  - farming gold,
  - buying tools,
  - researching tasks, and
  - overcoming monsters.

The goal is the Epic Win at the end of the year where the fully developed system is deployed to the end customer.

Junior Project Course Sequence Educational Objectives

The Junior Project course sequence is the first exposure for most software engineering technology students to large scale software engineering problems. Topics include software engineering process, scheduling, architecture, and teamwork. This is also the first time students work on a project spanning multiple quarters. In some instances, projects are a continuation from a previous year team effort. The junior project is a required sequence and must be completed in a single year. Once a project is started, the teams remain consistent throughout the year. The sequence is split up based on quarters with specific outcomes identified for each quarter.
<u>First Quarter – Software Process Management</u>

1. Describe and contrast different software engineering process models including the iterative process used in this class.
2. Understand the benefits and problems of teaming, describing qualities and processes of effective teams, and describing the role of teamwork in system design.
3. Describe and contrast potential project organizations and the team member roles within those organizations.
4. Create a Team Charter to articulate how the team will track, manage and communicate project progress, changes in scope, changes in design, and defects.
5. Assess risk, probability of the risk, triggers and formulate contingency plans.

6. Construct a statement of work with appropriate acceptance criteria.
7. Describe the relationship between Testing and Quality Assurance.
8. Describe the Quality assurance practices appropriate for each part of the development life cycle.
9. Create user based requirements and engineering requirements.
10. Describe traceability and be able to map a requirement through all project artifacts.
11. Describe different modeling techniques and where they apply.
12. Describe the different architectural views and assign them to parts of the life-cycle.
13. Asses risk and develop risk management plans.

Second Quarter – JP I

1. Use case modeling to refine requirements.
2. Create a use case specification including steps and scenarios.
3. Derive sequence models from use case scenarios.
4. Create initial class models from sequence diagrams, requirements and uses cases.
5. Describe the roles of team members including the Team Leader and Quality Assurance Manager.
6. Describe the role of functional analysis in use case analysis and requirements gathering and refinement.
7. Understand the role of the low fidelity User interface in the software development life-cycle.
8. Develop a project schedule including refining estimates and allocating resources.
9. Manage the project including collecting status, analyzing variances, planning and taking adaptive action and reporting status.

Third Quarter – JP II

1. Management of requirements and understand the importance of requirements in a large scale software project.
2. Use case analysis and understanding its place in the software development lifecycle.
3. Understanding of software development event sequences and how they fit in the software development lifecycle.
4. Application of class modeling and understand how it fits into the software development lifecycle.
5. Application of software system development task scheduling and development estimates.
6. Application of software development implementation (writing code) and understanding how it fits into the software development lifecycle.
7. Use of software system requirements and the mapping of those requirements into the software development lifecycle for effective impact analysis for change control.

MMORPG Course Elements

The goal of the course modification is to draw students into course material through application of scoring and organization of activities similar to ideas used in MMORPG. The following ten elements are considered the ingredients for any successful game[5].

1. Avatars
2. Three dimensional environment
3. Narrative Context (Epic Story)
4. Feedback
5. Reputations, Rank, and Levels
6. Marketplaces and Economies
7. Competition under explicit and enforced rules
8. Teams
9. Parallel Communication Systems
10. Time Pressure

The challenge is to adapt these elements to a standard three term, project based course. Items 1, 2, and 9 were of no consequence since the course is not being taught under any virtual gaming environment. Items 3 – 8 and 10 are directly supported through the structure of the course.

Narrative Context
Students were given a "first day on the job" lecture where the environment in which they would be working was explained. The projects were introduced, each project with a story behind the idea and a presented use scenario.

Feedback
There is no limit to the number of times a student could attempt completion of a task and submit the related deliverable for feedback. Each time a team or students submit a project artifact for points, comments are made on the artifact explaining how it can be improved.

Reputations, Rank and Levels
A leader board was maintained throughout the year showing the top three teams. This is a point of competition for team pride. The project is divided into levels so as a team progresses in implementation of the project, the team will advance to the next level. Each new level presents more complicated and difficult tasks as well as challenge tasks that bring external elements into the course content.

Marketplace and Economies
Individual students and teams were required to purchase tools from a virtual marketplace for use in completion of tasks. At each level, the marketplace is updated with additional tools that teams will require to complete the tasks related to the level. These tools required earning and then spending virtual gold. Teams also bartered tools and services, using the virtual gold as a currency. Projects were auctioned off where bidding was done in earned virtual gold.

Competition under explicit and enforced rules
Competition is enforced through the use of Experience Points as a measure of how well a team is doing, with the allocation of those experience points based on task performance. Technical challenges were set up in which teams compete. Rules govern team member behavior with specific consequences described for breaking course rules.

Teams
By nature of the class, students were assigned project teams for completion of their project. Project scope is larger than what most individual students could complete in a single academic year. Task size requires projects to be divided among team members and teams to coordinate for effective project integration and deployment.

Time Pressure
The three term course sequence establishes a three term deadline. Twice each quarter, individual students are presented with an assessment of how well they are doing in experience points as related to course outcome expectations. Each challenge task had a time limit for task engagement and task completion.

Level Definitions

The sequence objectives was spread throughout the three term sequence and presented in a sequential manner such that useful work could be done in completion of the final assigned project. The division was also done so students gained required skills for level completion as they progress through the sequence of levels. To define tasks for the sequential levels, topics and deliverables from previous years courses were sequenced and grouped based on key milestones in project development lifecycle. Each group of tasks and associated deliverable artifact was assigned a name to represent the level of experience required to take on the tasks. There is no sequencing of tasks within a level. Levels must be completed sequentially.

These groups were:

Level Name	**Tasks**
Novice Coder	1. Reading of Brooks, *The Mythical Man Month* and summarizing articles. 2. Reading of Martin, *Agile Software Development* and summarizing articles. 3. Structuring a development team and creating a team charter. 4. Bidding on a project to be worked on for the school year. 5. Researching offered projects to gain an understanding of potential systems to be built over the course of the year. Choosing a project to work on. 6. Gathering and cataloging system functional and non functional requirements. 7. Creation of an initial user interface prototype. 8. Generation of bi-weekly status reports.

Requirements Engineer	1. Researching and writing a system business case. 2. Building of a functional architecture description and UML model. 3. Creation of an initial Gantt chart for project management. 4. Definition of a software development process the development will attempt to follow throughout the development lifecycle for the system. 5. Creation of an enterprise level database for their system including defeat of the "Database Ogre". 6. Bi-weekly status report and Gantt chart update. 7. System demonstrations. 8. Technology demonstrations.
Systems Analyst	1. Use Case Specifications 2. Use Case Architecture Model and Model Description. 3. Early class model and logical architecture. 4. Requirements specification and defeat of the "Specification Sorceress". 5. Architectural Prototype presentation. 6. Requirements to Use Case map. 7. Bi-weekly status reports and Gantt chart update. 8. System demonstrations. 9. Technology demonstrations.
Software Designer	1. Use Case Model with complete associated sequence diagrams. 2. Logical architecture and description. 3. CRUD matrix. 4. Refined class models and descriptions. 5. Business case refinement and defeat of the "Business Troll". 6. System design presentation. 7. Bi-weekly status reports and Gantt chart update.
Software Architect	1. Refined class models with high and medium level priority requirements coverage. 2. Refined user interface and underlying implementation. 3. Initial component model. 4. Outsourcing exercise. 5. Initial deployed component artifacts based on defined interfaces. 6. Operational tests for deployed component artifacts and defeat of the "Test Dragon". 7. Component and test demonstration presentation.
Systems Architect	1. Completed class models and descriptions. 2. Refined component model with exposed interface dependencies. 3. Physical model and description. 4. Deployment description. 5. Alpha release deliverable - executable, installation scripts, and installation manual. 6. Alpha release test definitions, scripts, and execution. 7. Alpha release presentation and defeat of the "Presentation Boss". 8. Alpha release test completed and returned – done by a different team.
Software Engineer	1. Completed component model. 2. Alpha release cycle test results fixed and documented.

	3. Beta release deliverable - executable, installation scripts, and installation manual. 4. Beta release test definitions, scripts, and execution. 5. Beta release presentation. 6. Beta release test completed and returned – done by a different team. 7. Beta bugs fixed. 8. Final release done and defeat of the "Challenge of Final Doom".

The levels and associated tasks are introduced when a team levels up. The intent is to allow for a sense of discovery as the students work through tasks, revealing more information as they complete tasks.

Challenge tasks involve a professor and task external to the topics being directly addressed in the course but integral to successful project completion. These tasks are drawn from support topics directly related to the tasks in the level at which a team is currently working. For instance, all students, by the second term of the three term sequence, are either taking or have taken, technical report writing. The challenge related to technical report writing is to present their system specification to a specific writing professor for evaluation. The writing professor is instructed to be "brutal" in their assessment. The team must satisfy the writing professor in order to complete the challenge. The challenge tasks involved:

1. Database definition and implementation.
2. Software requirements specification technical writing
3. Business case definition and presentation
4. System test definition and execution
5. Public speaking and presentation

Experience Points and Knowledge Gold

The levels, tasks, and interaction of the course are supported by the use of experience points and knowledge gold. Experience points are used to gauge team and individual performance throughout the course sequence. Experience points ultimately translate into a grade at the end of the year. Knowledge gold is used for "purchase" of tools and exchange of services between teams. Both experience points and knowledge gold have rules that govern their acquisition and exchange.

Knowledge Gold

The course currency is called Knowledge Gold (KG). The following rules govern the use of currency in the three term sequence:

1. All students start with 0 KG at the beginning of the year.
2. KG is tracked by the instructor for each individual in the course. It is the student responsibility to make sure KG credit is properly recorded.

3. KG will be retained across quarters.
4. KG may be obtained by:
    a. Completing tasks as stipulated by the instructor.
    b. Asking good questions.
    c. Chapter write-ups.
    d. Research into a technical area related the course topics or topics directly related to a student's project.
    e. Productive feedback or thoughtful reflection about the course and process.
    f. Any other tasks that the instructor deems worthy.
5. KG may be lost by:
    a. Coming in late to class.
    b. Showing up late for or missing scheduled presentations.
    c. Being rude.
    d. Any other thing the instructor deems worthy of penalizing with a loss.
6. KG can be traded to other members of the class. All trade must be accompanied by a formal contract, signed by both parties, and then a copy given to the instructor.

    Examples exchanges include:

    a. Tasks such as programming or other project based help.
    b. Use of tools not possessed by the individual or group.
    c. Exchange of tools or other desirable items.

Experience Points

Experience Points (EPs) are awarded to individuals based on tasks completed toward the sound engineering of a software system. Team experience is the sum of the experience points possessed by all individuals on the team. As a team moves through the technical environment of the software development levels, it will obtain experience points by completing tasks from the level. The following rules govern the assignment and management of EPs:

1. The course instructor will track the experience points gained by individuals on a team for tasks completed.
2. It is up to teams and individuals to make sure EP accounting is correct for their cases.
3. EPs will be accumulated throughout the three term sequence. Grades will be assigned at the end of Spring term based on percentage of total available EPs for the year.
4. When a task or set of tasks is presented, the associated experience points will be shown for each task.
5. Some tasks are recurring; some are one time. Recurring tasks can be completed throughout the year; however, as a team moves up in levels, some of these recurring tasks will lose value in both awarded KG and EPs.
6. The amount of EPs awarded to the completion of a task is determined by the instructor at the time of evaluating the artifacts resulting from the task.
7. The team can choose how to divide awarded EPs among themselves. This must be communicated to the instructor at the time of EP recording. Conflicts will be mediated by the instructor. The decision of the instructor on these matters is binding and will not be further disputed.

8. If a team does a particularly bad job at completing a task, the instructor has the option to level the team down by stripping EPs from the team individuals. If this is done, the team must re-visit tasks completed in the new lower level to gain enough EPs to level back up.

Course Event Sequencing

At the start of the first term, the class was allowed to self form into student teams of three or four individuals. These teams remained consistent throughout the year. Team formation was done in the first week of the first term. Teams were then presented with a list of team roles. These were: team lead, technical lead, quality assurance lead, and lead programmer. Each team was required to submit a paper indicating what students were being assigned to which role. Teams were also required to come up with a team name and a slogan.

The projects were introduced in the second week of the first term. Students were informed they were all at the Novice Coder level. Project possibilities were presented allowing teams to discuss and barter for project topics. Bartering was done by exchanging Knowledge Gold for alliance during the bidding process. During the first three weeks, students were “farming gold” in anticipation of the project auction. This was a period of accelerated reading and writing.

In the third week of the first term, projects were auctioned off for knowledge gold. Once teams obtained their official project, work began in earnest. Teams that had started work on a project, yet did not obtain the specific project in the auction, sold their work to the winning team for knowledge gold. At this point in the first term, students were required to drive the sequencing of the course by requesting lecture topics revolving around required level tasks. Students also scheduled both individual and team based presentations. For any scheduled presentation outside required level presentations, a team was given knowledge gold and experience points. Once this position of the course was reached, the teams work level drove course topic sequencing.

Leveling up

Leveling up is the process of gaining enough skills through successful completion of a defined set of tasks to be presented a new set of tasks requiring more refined and new software engineering skills. The goal is to never present a task greater than what the students can achieve. Leveling is the act of moving from one experience level to the next. As a team moves up in levels, they will be presented with more difficult tasks related to the project they are working on. Each task relates to a different part of software development and engineering. Rules applied to leveling are:

1. A team must gain a specific level of experience points and complete all current level tasks in order to "level-up" to the next level of difficulty.
2. The experience points for leveling requirements will be calculated as an accumulation of experience points of all team members.
3. A team can only gain points for tasks at their current level or the levels below.

4. Levels will not be revealed before being achieved by a team.
5. The first team to level up will reveal the next level to the rest of the class.
6. The **first** team to level up will receive a bonus, determined at the time of leveling up by the instructor.
7. Tasks and tools will accompany the experience point level of the team. As a team levels up, they will receive the tasks for the higher level. Tools will be made available for purchase for the new level.
8. All tasks and tools from lower levels will be available at higher levels. Some tasks are recurring, and Experience Points and Knowledge Gold can be continuously garnered from these tasks.
9. It is possible for an individual or team to level down if their performance on a task related to the their level is particularly bad. When this happens, the team individuals will lose EPs that place them in the lower level.
10. A team will only level up if their total accumulation of experience points is at the 60% level of all available accumulated experience points for their level. If a team is not at the required accumulation of experience points, they will be required to either do additional tasks or re-work some of the level tasks where they performed less than optimal.
11. Level based presentations can be done only once.

MMORPG Course Grading

Grading of the course is based on percentage of experience points and all required deliverables obtainable at the level of Software Engineer. During the first two terms, all students in the course were given an in-progress grade. At the end of the third term, 90% or above of obtainable experience points is equivalent to an A, 80 – 89% of obtainable experience points is equivalent to a B, 70 – 79% of obtainable experience points is equivalent to a C, 60 – 69% of obtainable experience points is equivalent to a D, and below 59% is an equivalent F. The grading for this course is high stakes for the students since the grade received will represent three courses and a grade of a D or an F will require the student to take the entire sequence over.

Each deliverable in the sequence of tasks is graded by the instructor. Points are assigned to each deliverable artifact by use of a scoring rubric. Students are given the scoring rubric when they obtain the level. Each time a team turns in a set of artifacts for a task, the artifacts are scored and the completed rubric is returned to the team. The team can then choose to work further on the task and re-submit the artifacts, or it they can choose to accept the score and move on to another task.

Presentations are graded by the class and instructor as a whole based on a grading rubric. The presentation rubric is distributed when a team levels up to a level with a formalized presentation. Each student in the course is required to attend a scheduled formal presentation. Excuses must be cleared prior to the day of the presentation. During the presentation, students, instructor, and any

interested visitors use the provided rubric to score the presentation. The scores are then averaged with the instructor and any special visitor scores given more weight than the student scores.

Project Definitions for the 2010-2011 Academic Year

Projects were defined by the course instructor. In all cases, the projects are real-world, cross-discipline projects where project completion will result in system deployment and use by the project sponsoring client. This gives the project a real-life stake holder with keen interest in project success. All projects have enterprise level components.

Robot Navigation – Manufacturing Robotics
In a robotics-based manufacturing environment, a manufacturing floor will be split up into areas of robotics activities called work cells. A robotics work cell is a grouping of robots that are tasked with creation of a particular part of an end product. To affect full product assembly, work cells need to be fed incoming parts for use in assembly, and completed parts need to be taken away. To supply work cells and remove completed assemblies from work cells, a robot will move parts between areas of the manufacturing floor. This project involves interfacing of a robot with a ceiling mounted camera for the purpose of navigating a parts delivery robot around on a manufacturing floor. This project will involve:

- Image acquisition and processing from a digital camera.
- Path finding from the imaging system.
- Robot programming for navigation on a shop floor.
- Robot building and possible augmentation.
- Potential embedded system programming.
- Client/Server wireless communication

Electric Vehicle Data Telemetry and Tracking
An electric vehicle company is working with the Junior Project class on a project for data acquisition and dissemination from the electric delivery truck being built for initial sale in the Brazilian market. The system being created is a remote telemetry system based on CDMA or GSM cellular technologies. The system will interface with vehicle sensors and the control system. The overall goal of the system is to acquire real-time data from the vehicle as it travels. The data stream will then be sent to a centralized web site where it will be displayed and archived for later analysis. This project will involve the following:

- Interfacing with a real-time control system.
- Data acquisition, processing, and streaming through a wireless network.
- The design and implementation of web technologies for display and end user analysis of the data.
- Use of GPS technologies for vehicle location tracking.

Catheter Lab Simulation

A Cath is a “catheter” used for interventional radiology for treatment of cardiac emergencies. The catheter is directed through an artery to a region of interest to deliver medicine for relief of clogged arteries. The Cath Lab at our university has a camera and workstation to act as a “C Arm” or “U Arm”. The project is to build software and adapt an X, Y, Z accelerometer to the camera for to simulation of the Cath Lab procedure behavior. This also requires the application of a foot pedal to initiate image playback. The project will require the following:

- Interfacing to a real-time accelerometer.
- Real-time image capture, manipulation, and playback from a digital camera.
- JPG image capture, manipulation, display and storage.
- DICOM image capture, manipulation, display and storage.
- Touch screen tablet interface.

Batch DICOM Image Writer

Medical Imaging creates large volumes of digital images. These images are stored in a centralized database called a Picture Archival Communication System (PACS). The medical imaging department has purchased a DVD batch writer. Currently there are no inexpensive tools allowing the retrieval of DICOM based images from a PACS system, viewing and manipulation of those images, addition of annotation, then writing of the images onto a DVD in DICOM standard format. This project will involve:

- Creation of a DICOM viewer for acquiring images from the OIT PACS and displaying those images in a UI.
- Allowing annotation to be added to the images in the viewer.
- Writing the images to a DVD in DICOM standard Part 10 format.

Tracking Robot

Last year, progress was made on a robot intended to act as a "watch dog" where the robot would sit idle in a room. Upon sensing some disturbance in the room, the robot would enter a search mode, looking for some individual to acquire as a target. Upon acquiring the target, the robot is to follow the target and take pictures, sending those pictures to a web server. The current robot will acquire a target but not track it. To track a target, the Hume tracking algorithm must be implemented. This project will involve:

- Refinement of an existing robotics platform.
- Application of numerical analysis constructs.
- Use of Microsoft Robotics Studio (MSRS) to create services

MMORPG Course Assessment

End results of the course were mixed. There were positive and negative aspects of student engagement. Overall, the outcome was similar to that of the course as done in a standard lecture/lab venue where tasks were hard scheduled and dates were set. Students gained experience and knowledge as required by the three term sequence. Students generally maintained

a higher level of interest in course activities, but it was easier to "lose the group" during times of high external commitment (such as midterms) or times of low perceived activity. A summary comparison is offered in Table 1. Comparative analysis is based on the previous year software engineering sequence as compared to the sequence offered under the MMORPG grading model.

Table 1 – Comparison of course outcomes form standard grading vs. MMORPG style grading.

	Standard Lecture/Lab Sequencing	**MMORPG Style Sequencing**
Student Attitude	Students typically complain about the heavy workload related to leaning a new set of theories and skills combined with the necessity of practicing and producing a working product.	Students did not recognize the workload as being excessive and retained a positive attitude throughout the sequence. Students felt they could better control the work load to meet demands of other courses.
Group Dynamics	Dysfunctional groups typically emerge toward the end of the second term of the three term sequence. At this point, problems are so engrained in group dynamics that recover and success is difficult.	Group problems came to the surface early in the sequence. This allowed groups to address issues and develop good working relationships. Problems addressed early did not come up again.
Group Performance	Poorly performing groups remained poor. The specific deadlines caused early demoralization in groups with trouble in working relationships.	Low performing groups maintained the ability to advance and did. In the first year of this style grading, two of three low performing groups transformed themselves into mid level and high performers after overcoming rocky beginnings. These groups accelerated at the middle of the second term and continued positive advancement.
Reading	~50% of the students would read 100% of the assigned reading. No students read beyond assigned reading in software engineering.	~ 10% of the students extended their reading on software engineering to include current BLOGs. ~15% read far more than the suggested reading. ~35% read at the same as the standard course. ~30% read below the suggested reading. ~10% did no assigned reading at all.
Writing	~40% of students wrote the assigned number of essays on software engineering.	~45% of students wrote more than the standard assigned number of software engineering essays.

	~50% of students wrote below the assigned number of essays on software engineering. ~10% of students turned in no writing at all. ~50% of students showed noticeable increase in technical writing proficiency during the three term sequence.	~20% of students wrote the standard assigned number of software engineering essays. ~35% of students wrote less than the standard assigned number of software essays. ~75% of students showed noticeable increase in technical writing proficiency during the three term sequence.
Work Artifact Quality	Work quality typically reflected the desired grade of the students. Quality was consistent throughout the three term sequence.	Work quality was consistently high for students in high performing groups. Mid level performing groups would have varying levels of work quality dependent on interest in gaining an edge on competing groups. Some of the highest quality work was produced by mid level performing groups.
Internalizing of Material	Material presented early was typically forgotten by the end of the three term sequence. Students tended to attempt rote memorization of terminology and concepts specific to scheduled activities and deliverable project artifacts.	Material presented early in the three term sequence was well understood by the end of the year. The later the topics were introduced, the less students seemed to have software engineering concepts at direct recall when interviewed.
Skill Attainment	Students gained satisfactory skills for advancement to the senior project sequence. Very few students would stand out of the crowd.	Attainment of skills related to items introduced early was very high. Skills related to items introduced later in the sequence was lower; however, still deemed acceptable for entry into the senior project sequence.

Negative Outcome

Keeping track of knowledge gold and experience points proved to be a challenge. Students ended up submitting more work for review, then re-submitting to gain more points and gold. This required a careful consideration of how many points they had previously received for a particular artifact and how much the modification was worth. The correlation of knowledge gold and experience points also required careful attention.

Students were constantly “gaming the system”, finding loopholes in rules for task completion, task scoring, and resource exchange. This required constant vigilance over how the class was intermingling and trying to uncover areas of problem before teams discovered and exploited the

loopholes for their benefit. Although increased student involvement is a good outcome, in this case, the involvement was directed at a tangential artifact of the grading system. Energy spent here was negatively traded for time on required software engineering knowledge and skills.

The instructor is required to manage a small economy where activity in the class is dependent on the health of the course economy. This requires understanding of the use of knowledge gold and experience points for the purpose of applying incentives to get the student to perform work toward the end goal of the project. This management job turned out to be time consuming. Course economic incentives were constantly being adjusted. Each time an adjustment was made, students had to be informed, and the class website needed to be adjusted.

By the end of the first term, low performing teams were far behind the high performers. This made low performers get course topics introduced long before they were ready to apply the knowledge. This gap increased over time increasing difficulty for low performers. At the end of the second term, despite constant help sessions, the lowest performing teams had fallen so far behind there was no hope in completion of required tasks. Ultimately, one team had to be removed from the MMORPG system in an attempt to get these students the required course content before the end of the year.

Positive Outcome

The structure of the grading system seemed to remove the stigma of doing things wrong, then asking how to do them correctly. Several key concepts in software engineering were visited throughout the three term sequence based on desires of students during directed discussions. Students would request lecture on topics when relevant to their projects, thus forcing repetition and different viewpoints for other course participants. Students never complained about topic repetition. The requesting teams' questions seemed to bring relevance to other teams.

It became apparent early in the three term sequence which teams were going to perform well and which teams were going to have trouble. This made it possible to target the poorly performing teams to try to get work up to an acceptable level. More importantly, it became apparent to the teams themselves which team members were not going to perform to the level expected by the team. This spurred teams to come to the course instructor early for advice on dealing with these problems.

On the whole, students spent more time reading required texts and writing summaries of chapters and articles in the texts. This is largely due to the use of these activities to bolster an individual and team's accumulation of experience points and knowledge gold. The reading of articles and writing of synopsis was seen as "farming gold" by the student population. Students also showed a higher level of motivation in the understanding of reading material and the application of the concepts in the reading material to the project with which the team was working. Several students remarked on how much they enjoyed the articles in The Mythical Man Month by Fred

Brooks. Even though this is essential reading for software engineering students, it is the first time students have openly stated their appreciation of this reading.

The general attitude of students toward the topics presented in the course sequence was much more positive than in the past. Students were not being forced to complete tasks that they did not see as beneficial to their understanding of software engineering; they were allowed to request information when they saw it necessary for completion of tasks they had chosen to take on. This difference in approach made the curriculum and curriculum delivery under control of the students, forcing them to take ownership of when and how topics were introduced and approached.

Challenge tasks brought external skills to light in the arena of software engineering. In no case did any student complain about the outcome of the challenge task; however, students did complain and question relevance going into the challenge tasks. The challenge tasks made the software engineering exercises real, bringing outside expertise to light in the completing of a defined software engineering task.

Recommended Changes

Before offering the Junior Project sequence under the MMORPG points system, the following recommendations will be put in place:

- Creation of a "smarter" grade book.

  As mentioned earlier, tracking of experience points and correlated knowledge gold was a challenge. The development of a spread sheet grading system would ease this burden greatly. Adding logic to the grade book template in the form: If cell (x,y) has and entry, add "z" to "page 2" row "x". The addition of this simple logic would allow the course professor to track only tasks completion. The grade book system must also be able to track when a task was started and ideally indicating a time-out based on a maximum task duration.

- Adaptation of policies related to stragglers and the task trial/error cycle.

  In this first attempt at applying the MMORPG style points system to the project based course sequence, there were no checks in place to handle the incredibly slow teams or the teams striving for 100% perfection. This allowed teams to get hopelessly behind or spend too much time trying to achieve perfection. In the next iteration of this style grading, each task will be assigned a minimum start time and a maximum duration time. If a team falls behind the minimum task time, it will forfeit access to the MMORPG grading system and fall under a standard schedule.

- Require periodic project demonstrations throughout the entire sequence.

  Students focused directly on perceived tasks at hand. If some form of code based deliverable was not immediately required, no work would be done on the technical aspects of the system being constructed. Bi-weekly demonstrations proved an effective method to keep students focused on the ultimate project artifact – a working system.

Conclusions

The MMORPG grading method had positive and negative outcomes. The positive aspects revolved around the high performing groups and their drive to compete in a technical project environment. These teams embraced the course content and raced toward the finish from day one. The grading and self guided mechanism encouraged these teams to push hard and reap rewards of fame amongst their peers. Never underestimate the power of bragging rights.

Negative aspects revolved around the low performing teams and the freedom they were given to fail miserably. Being a mandatory sequence in the curriculum where passing with a C or better is required to move into senior year courses, getting a D or F in this course would, at best, set a student back an entire year in the curriculum. Low performing teams are not necessarily comprised of low performing students. These teams need help, not reprimand. This first round of MMORPG style grading did not support the low performing conditions well.

Ideally, this style of grading could be used as an initial part of a longer sequence on software engineering. By the end of the first term, it was clear which teams would be high performers and which teams would fall behind. This highly competitive grading scheme would serve well as a litmus test to divide teams into honors based or standard tracks. After the first term of competition, low performing teams would be placed into a class where standard schedule/lecture/project assignments would be used. High performing teams moved into a class supported by the MMORPG style grading. The MMORPG grading style proved very beneficial for highly motivated project teams. Keeping these teams moving at a self regulated pace helps these individual realize their full potential and optimizes their learning experience.

---

[1] Total Engagement, Using Games and Virtual Worlds to Change the way People Work and Businesses Compete, Harvard Business Press, 2009.
[2] Pew Internet and American Life Project, "Teens, Video Games, and Civics", Pew Research Center, September 16, 2008
[3] Jane McGonigal, Gaming Can Make a Better World, TED2010, Technology Entertainment Design, Long Beach, CA, February 2010
[4] Gaming the System *What Higher Education Can Learn from Multiplayer Online Worlds*, J. C. Herz, 2009, p. 173
[5] Total Engagement, Using Games and Virtual Worlds to Change the way People Work and Businesses Compete, Harvard Business Press, 2009, pp 63-90