Teaching Software Engineering Through Game Design

Kajal Claypool
Computer Science Department
University of Massachusetts
1 University Avenue
Lowell, MA, USA
kajal@cs.uml.edu

Mark Claypool
Computer Science Department
Worcester Polytechnic Institute
100 Institute Road
Worcester, MA, USA
claypool@cs.uml.edu

ABSTRACT

Many projects currently used in Software Engineering curricula lack both the "fun factor" needed to engage students, as well as the practical realism of engineering projects that include other computer science disciplines such as Software Engineering, Networks, or Human Computer Interaction. This paper reports on our endeavor to enhance interest and retention in an existing Software Engineering curriculum through the use of computer game-based projects. Specifically, a set of game-centric, project-based modules have been developed that enable students to: (1) actively participate in the different phases of the software lifecycle taking a single project from requirement elicitation to testing and maintenance; (2) expose students to real issues in project and team management over the course of a 2-semester project; and at the same time (3) introduce students to the different aspects of computer game design. Preliminary results suggest the merits of our approach, showing improved class participation and performance.

Categories and Subject Descriptors

K.3.2 [**Computers and Education**]: Computer and Information Science Education,Computer Science Education; D.2.0 [**Software Engineering**]: General

Keywords

Computer Games, Game Design, Game Engineering, Software Engineering

1. INTRODUCTION

Software Engineering courses often teach about coding and managing large projects – projects that integrate many different Computer Science disciplines under one umbrella. A good Software Engineering project should ideally bring together not just basic software engineering principles, but also knowledge and coding acquired in areas such as algorithms, user interfaces, databases and networking. A good project should also enable students to shift between low-level tasks such as algorithm implementation to high-level tasks such as system and object-oriented design.

Computer games provide a rich opportunity to bring these ideals together in a traditional Software Engineering course. Game development combines techniques from a wide spectrum of Computer Science including Software Engineering, Graphics, Artificial Intelligence, Networking and Human Computer Interaction. Game production requires and emphasizes the need for students to develop skill sets ranging from high-level design techniques to low-level implementation. The visual and interactive nature of computer games that attracts millions of game players world-wide [6] can also deeply engage student interest, and the end results are often clear-cut and enjoyable. Moreover, while games often involve creating real-time simulations of an artificial world, the skills and techniques used in its production are transferable to a wide-range of simulations of other systems. On the economic front, many students want to get jobs as game developers and are thus further motivated by the prospect of developing a game. A completed computer game is an impressive addition to a student portfolio for showing to potential employers, whatever the nature of the job, as it demonstrates both a breadth of knowledge in many Computer Science disciplines and an in-depth knowledge of basic Software Engineering principles and software lifecycles.

A primary goal of this work is to provide a better way of teaching Software Engineering by developing a curriculum that integrates computer game engineering with Software Engineering in a project-based learning environment. A side benefit of this work is to develop and implement a curriculum that meets the demands and interests of the next generation of Software Engineers - the *Game Engineers*. Computer games have become a major industry and are one of the fastest growing application markets in the world. The U.S computer and video game software sales exhibited a doubling of industry software sales in 1996 to $7 billion in 2003 [6]. To manage the large projects that a major game release requires, the gaming industry uses a mixture of techniques and concepts borrowed from software development, the movie industry, and traditional games. Although this has been effective, as seen from the games on the market, there is a growing concern within the gaming industry that a developed design and engineering discipline is lacking [14, 3], especially one developed to support multidisciplinary groups that in practice create the games. The focus of this work is on a curriculum that not only targets training game engineers by exposing them to the fundamentals of Software

ITiCSE '05 June 27-29, 2005, Monte De Caparica, Portugal

Engineering but also on applying Software Engineering skills to address the challenge of game engineering.

This paper reports on progress towards the development of a set of game-centric, project-based modules for *Computer Games in Software Engineering* courses. From a Software Engineering perspective, the specific goals of the game-centric, project-based modules are: (1) to have students actively participate in the different phases of the software lifecycle taking a single project from requirement elicitation to testing and maintenance; and (2) to expose students to real issues of project and team management over the course of a 2-semester project. From a game engineering perspective, the objectives of the enhanced Software Engineering curriculum are to: (1) introduce students to the different aspects of computer game design; and (2) help students develop the skill sets necessary to identify common patterns in games to enhance re-usability of game software.

The first phase of our approach is near completion, resulting in an integration of computer game design with a Software Engineering course. The modules for this course have been implemented, along with additional course material including syllabus, slides, projects, and other course materials specific to game design in Software Engineering. Preliminary evaluation suggests merits to our approach, with increased enrollment, better student performance and increased student participation in all aspects of the course. As an additional contribution to the academic community, all the modules produced are available on the Web.

The rest of this paper is laid out as follows: Section 2 presents the plan for integrating computer games into Software Engineering; Section 3 outlines a sample set of projects used in the course; Section 4 describes our progress thus far and discusses preliminary results; and Section 5 summarizes our conclusions and mentions possible future work.

2. MODULES

This section presents game-based project modules that integrate computer games into an existing Software Engineering curricula. The approach combines aspects of game engineering with Software Engineering with the goal of providing a disciplined approach to engineering a game while emphasizing core Software Engineering principles. We break the course material down into a set of *modules*, where each module represents a Software Engineering concept and includes a set of lecture slides, handouts, and where appropriate projects. Detailed content and supporting material for each module can be found online [2]. A single overarching project, a computer game, with milestones corresponding to the individual modules are developed to emphasize both the overall Software Engineering of a large project and the technical content of each module. A sample set of projects is outlined in Section 3.

Module 1: Introduction. This module provides an overview of the general topics of software and game engineering, the Unified Modeling Language (UML) [4], and the Eclipse [9] development environment. A key focus of this module is in providing a common terminology for both software and game engineering. Students are introduced to UML as a key design technology and to the tool support offered by Eclipse and its plugins.

Module 2: Development Lifecycles. This module provides an overview of the different software development lifecycles and contrasts it to the game development lifecycle. Topics in this module include the study of different software process models, techniques for both static and dynamic modeling, the study of game development models, and the role of software process models in game development. Game development models and the interaction of software development models with game development models is an active area of research in both academia and industry. The material developed for this unit of the module thus uses published research papers.

Module 3: Project and Team Management. This module provides a formalization of project scheduling and team management. This is a critical phase to ensure on-time delivery and provide essential risk management. Topics in this module include project planning, effort estimation, risk management, interaction between the development models and introduction to tools for tracking project schedules and teams. Although the fundamentals of this module are directly applicable to game engineering, it should be recognized that the multi-disciplinary teams typical in game development introduce a higher risk and require additional time in team management.

Module 4: Requirement Elicitation. Requirements elicitation is the formal process that results in the specification of the system that the client understands. In game engineering, a parallel to requirements elicitation is the development of the *game idea* that goes on to become the *Pitch* document. This module provides an overview of requirements elicitation and game idea development. Topics covered as part of this module are techniques for documenting elicitation, the definition of meaningful play, storyboarding techniques to represent the game idea, the requirements elicitation and pitch documents. Special emphasis has been made on the consolidation of these two processes into an integrated process that uses adapted software engineering techniques to provide a formalization of the game idea. The main output for this module is the development of a *Game Concept Worksheet* for the class project – a game being developed by individual teams of 2 to 3 students. The worksheet combines the key features of both a typical requirement elicitation document and a game's high pitch document.

Module 5: Introduction to Game Design. This module provides in-depth exploration at the key elements of game design [11] ranging from issues and guidelines for developing the game setting and game world, to storytelling and narratives, to designing a rich game play environment that preserves the internal economy of games and provides a balanced game. Topics in this module segue into a discussion of the common *design patterns* for different genre of games providing a set of features and requirements typical for a given genre. The main output for this module is the development of a game *prototype*, based on the individual team's *Game Concept Worksheet*, using the `Game Maker` [10] game prototyping tool.

Module 6: Requirement Analysis. Requirement analysis results in a model of the system that aims to be correct, complete, consistent, and unambiguous. In game development, this phase corresponds to the process of game design. While this module provides an overview of the traditional requirement analysis phase, there is additional emphasis on the integration of the key game elements, especially the game prototype developed in the previous module, to drive the requirement analysis. Topics covered as part of this module include techniques to focus on the identification of objects, their behavior, their relationships, their classification, and their organization. Additionally, to accommodate game design, extensions to the requirements analysis phase are examined that accommodate the semiotics of a game resulting in *meaningful play* [13]. The key output of this module is the development of a modified *Requirements Analysis Document* – a *Game Treatment Document* that brings together the key aspects of game design and requirements analysis under one umbrella.

Module 7: System Design. System design is the transformation of an analysis model into a system design model. During system design, developers define the design goals of the project and decompose the system into smaller subsystems that can be realized by individual teams. This module covers the basic steps of system design and provides adaptations to this process that take into account system aspects particular to the game development domain, such as the typical subsystems common to most games. Topics in this module include strategies for building the system, selection of the off-the-shelf components, as well as an in-depth coverage of architectural patterns [1] that can be used as guidelines for system decomposition. The materials developed for this module include a description of game genre design patterns, a topic currently being researched by both academia and industry [8, 12, 7]. The output of this module is two-fold. One is the development of the *System Design Document* that pays special attention to the identification of design goals in the game domain, and the subsystem decomposition and analysis that uses common game architectural patterns and subsystems in mind. A second output is the development of a prototype of a subsystem common to most 2D games.

Module 8: Object Design. Object design closes the gap between the application objects and the off-the-shelf components by identifying additional solution objects and refining existing objects. The identification of existing patterns and components is central to this problem-solving process. Topics in this module discuss these building blocks and the activities related to them, provide an overview of object design and focus on reuse, that is the application of design patterns [5], and discuss the techniques for identifying missing interfaces and functionality. While an overview of the main design patterns [5] is done, special emphasis is on the design patterns that have been proven to be more useful for the development of games. This module also provides coverage of game design patterns [8, 12, 7] to help students identify the collusion patterns and modular the rules for the artificial intelligence engine used in the game. Materials from this module include a categorization and description of the game design patterns. The output for this module is a complete object design of a subsystem identified in the previous module, with the emphasis to provide students with an opportunity to practice their skill sets for developing reusable design based on commonly used design patterns. Students also continue to refine and add to their prototype developed in the previous module based on the object design conducted in this phase.

Module 9: Implementation. This module introduces the core of code development coupled with game programming. Topics included in this module are mapping design to code, which includes formal methods for mapping associations and collections to code, programming platforms, the choice of programming language for game development, the use of multi-languages in game development, and techniques for re-engineering, re-factoring and code optimization. Code quality, code reuse and code refactoring are the special emphasis topics for this module. Students are exposed to the idea of full and partial code reviews and on the utility of coding standards. Output from this module is the development of other subsystems in each team's game. Based on the principles of reusability and the set of subsystems common to most games, each team develops a distinct subsystem as a general component for a game. For example, every 2D game requires a subsystem that can describe and load the game map. The `Map` subsystem developed by one team is utilized by all other teams in the class.

Module 10: Testing. Testing is the process of finding differences between the expected behavior specified by system models and the observed behavior of the implemented system. This module introduces concepts of fault, erroneous state, failure, and test, as well as the testing activities that result in the plan, design, and execution of tests. Topics in this module include functional testing techniques, selected testing techniques for nonfunctional requirements such as usability testing. To incorporate testing for games additional testing techniques are introduced to handle stress and playability testing. Students are also introduced to JUnit, a testing framework incorporated into the Eclipse development environment that enables the description of test cases, test suites, and provides an interface for the execution of the test cases.

3. EXAMPLE PROJECTS

This section briefly describes a sample set of projects that were developed to support the game-based project modules presented in Section 2. The projects all build upon each other as part of a working computer game. Project writeups and some project samples can be found online [2].

Project 1: Tools Background. Students are given a range of Java exercises to be coded in Eclipse in order to provide exposure to Eclipse and to increase proficiency with Java. Students also work through the *Maze Game Tutorial*, available with Game Maker [10], and extend the maze game minimally to demonstrate their grasp of basic Game Maker features.

Project 2: The Game Pitch. Students form a team of two to three students and elect a team manager that schedules meetings and sets up the project schedule. Each team develops a *Game Pitch* document describing the game idea and makes an initial sales pitch for their game.

Project 3: The Game Treatment. Based on the pitch, each team defines an initial *Game Treatment* document as part of requirement elicitation that describes the features of the game, including the game challenges, modes of play, character avatars, events and levels. The treatment also identifies non-functional requirements for the game, including character representation, and audio and visual elements. More traditional software engineering requirements such as the usability, reliability, and supportability of the game software are also described.

Project 4: The Game Prototype. From the treatment, each team categorizes their game's functional requirements into sets of *core*, *required* and *desired* features. Students then implement a prototype of their game using Game Maker based on the core requirements.

Project 5: Requirement Analysis. Each team constructs both an object model as well as a dynamic model based on the game prototype. Versions of the game interface are developed, as well as screen mockups and navigational paths.

Project 6: System Design. Each team identifies three key design goals for their game and decomposes their game system into smaller subsystems based on these goals. As part of this decomposition, students select an architecture style that can be applied to their overall game system design. The outcome of this project is the *System Design* document written based on a provided template.

Project 7: Object Design. Each team builds a detailed design of the *TileMapRenderer* – a subsystem that loads and renders a 2D map – based on object design principles. The outcome of this project is a UML class diagram representing the design of the subsystem.

Project 8: Implementation. Each team member implements a randomly assigned sub-module from the TileMapRenderer subsystem. In addition to the code for the working sub-module, each student provides a JUnit-conforming test harness and test case suite.

Project 9: Integration and Testing. Students integrate their individual sub-modules to construct, and subsequently test, the TileMapRenderer subsystem. The outcome is a working subsystem, together with a JUnit conforming test harness and test suite.

4. PRELIMINARY RESULTS

Game design has been fully integrated a Fall 2004 Software Engineering course (91.411 Software Engineering I), at the University of Massachusetts, Lowell. This integration includes design and implementation of modules for the course, as well as other supporting material such as syllabus, slides, projects, tools, and text book selections.

Preliminary evaluation has been done comparing the impact of the new game design modules on Software Engineering compared with traditional Software Engineering as represented by previous course offerings. Objective measures of performance include enrollment, class dropout rate, and exam scores. These quantitative measures are supplemented by subjective comments on the game-specific modules.

Table 4 depicts a summary of the objective results, where "Traditional SE" represents results from the previous offering of 91.411 Software Engineering I (Fall 2003) and "Game-Enhanced SE" represents preliminary results from the current, game-enhanced offering of 91.411 Software Engineering I (Fall 2004).

	Traditional SE	Game-Enhanced SE
Enrollment	7	22
Drop-outs	2	1
Average GPA	3.14	3.45
A's	2	12
B's	4	8
C's	1	2

The enrollment for the game-enhanced Software Engineering course is significantly higher than the enrollment for the traditional Software Engineering course, as well as other elective courses offered at the same time (8 being the average enrollment for all elective courses in Fall 2004). Before the course was offered, the intent to focus the Software Engineering projects on computer games was made known to potential students and a preliminary syllabus was available for students to peruse before enrolling in the class.

The breakdown of grades for students in the game-enhanced Software Engineering course is significantly different compared to the breakdown of grades in the traditional Software Engineering course, with the game-enhanced Software Engineering course having a higher distribution of A grades. These grades reflect significantly better student understanding of class material and noticeable improvements in the software engineering quality of the projects.

Surveys conducted during the course have provided us with subjective comments on the modules, primarily in the form of free-form comments. Here a small excerpt of some of the comments are provided that summarize some of the opinions voiced in the use of game-enhanced modules for Software Engineering:

> "Doing game projects is really exciting, my friends are envious and want to take this. When are you offering this course next?"

> "This course is sweet! Our group is already thinking about other games that we can develop after the course is done."

> "The game software engineering course was a good experience for me. It made the learning of the somewhat dry material :-) much more fun!"

5. CONCLUSIONS AND FUTURE WORK

There is a growing demand for robust, re-usable software that merges interdisciplinary Computer Science topics. This demand drives the need for effective methods of teaching of Software Engineering to tomorrow's application developers. The growing popularity of computer games coupled with the Computer Science sophistication required to build today's entertainment applications, presents and opportunity to use computer games as a means to better train Software Engineers. Game-centric, project-based modules can be used to illustrate all aspects of the software lifecycle, tapping into a broad range of Computer Science disciplines as is required to build modern applications while enticing students to grasp and apply Software Engineering to such disciplines by using games as a powerful motivator.

This paper presents initial work towards the goal of more effective Software Engineering education, describing some details of the implementation of a game-centric Software Engineering course. The focus has been on modules that allow for a hands-on practice of Software Engineering theory, where the sum of the modules culminates in a working computer game that clearly illustrates successful Software Engineering and provides students with particular satisfaction.

Preliminary evaluation suggests the merits of the approach. In the initial offering, class enrollment was up, drop-outs were down, grades had noticeably improved, and subjective survey comments from students suggested a greater interest in Software Engineering as a whole.

Future work includes more extensive evaluation, including qualitative and long-term evaluation. One hypothesis is that a project class which engages students will motivate them in other Computer Science courses. Measuring student grades in Computer Science pre-class and post-class could determine if the game-centric Software Engineering class engaged students sufficiently to improve their performance in later classes.

Future work could be to explore other opportunities for game-enhanced versions of other CS courses, with the intention to motivate and thereby improving the CS education of undergraduate students. Such game-enhancement could happen horizontally, by making game-centric versions of other senior level courses such as Artificial Intelligence or Computer Networks. Or, game-enhancement could be spread vertically, motivating students in early introductory courses right up through senior level courses.

6. REFERENCES

[1] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal. *Pattern Oriented Software Architecture, Volume 1: A System of Patterns*. John Wiley and Sons, 1996.

[2] K. Claypool and M. Claypool. http://www.cs.uml.edu/~kajal/research/pubs/-itisce05-SE-Games.

[3] G. Costikyan. I have no words and i must design. In *Conference Proceedings of Computer Games and Digital Cultures*, pages 9–33, 2002.

[4] M. Fowler and K. Scott. *UML Distilled Second Edition "A Brief Guide to the Standard Object Modeling Language*. Addison-Wesley, 2000.

[5] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Publishing Company, Reading, Massachusetts, 1995.

[6] IDSA. Essential facts about computer and video game industry - 2003 sales, demographics and usage data. http://www.idsa.com, 2003.

[7] B. Kreimeier. The case for game design patterns. http://www.gamasutra.com/features/20020313/-kreimeier_01.htm.

[8] S. Lundgren and S. Bjork. Game mechanics: Describing computer-augmented games in terms of interaction. In *Proceedings of TIDSE*, Nov. 2003.

[9] I. Object Technology International. Eclipse Platform Technology Overview. http://www.eclipse.org, 2003.

[10] M. Overmars. Game Maker. http://www.cs.uu.nl/people/markov/gmaker/, 2004.

[11] A. Rollings and E. Adams. *On Game Design*. New Riders Publishing, 2003.

[12] S. L. S. Bjork and J. Holopainen. Game design patterns. In *Proceedings of Digital Games Research*, Nov. 2003.

[13] K. Salen and E. Zimmerman. *Rules of Play - Game Design Fundamentals*. MIT Press, 2004.

[14] W. Spector. Remodeling RPGs for the new millennium. http://www.gamasutra.com/features/-game_design/19990115/remodeling_01.htm, Jan. 1999.