

Teaching an Object-Oriented Software Development Lifecycle in Undergraduate Software Engineering Education

M. Brian Blake and Todd Cornett
Department of Computer Science
Georgetown University
Washington, DC 20057
{blakeb, cornett}@cs.georgetown.edu

Abstract

In some cases, real-world application of software engineering concepts does not effectively map with current undergraduate curriculums. Typically, a student's first "hands-on" experience working on large-scale software development projects is via an intern position or their first full-time position. However, prior exposure to the corporate project environment would greatly improve a student's performance in industry. In order to develop students for successful careers in software engineering, specifically for software development, they must not only be immersed in the software development lifecycle and paradigms, but also in the workings of large project teams. Currently, most undergraduate software engineering courses are taught by presenting the concepts and methodologies and assigning fragmented 3-4 person group projects. In the Department of Computer Science at Georgetown University, a two-course approach to undergraduate software engineering education has been developed that incorporates the practical application of coursework in a large team setting. The first course presents a firm software design basis, while the second course demonstrates corporate-level software engineering concepts with a semester-long software development simulation where the entire class is the development team. This paper presents the experiences from offering this software engineering simulation approach.

1. Introduction

Software engineering is a broad field that encompasses many different areas. With the inception of Internet technologies, software engineering has become associated with a vast majority of areas, such as system development, parallel processing, database engineering and data management, network and distributed system security, software architecture and design, programming languages, performance modeling, graphical user interface design and others. It is impractical to think that a generic software engineering curriculum can prepare undergraduates for the intricacies of all these areas in addition to future upcoming areas. However, there are certain skills that can be taught that will allow students to deal with these multi-discipline areas in the software engineering environment

In undergraduate software engineering education, there is a debate between vertical and horizontal specialization [5][10]. This approach is toward neither a horizontal or vertical specialization, rather toward the understanding of the collaboration between multiple specialized fields in general. This approach teaches the cross-discipline collaboration, by creating an environment where students are assigned specialized software engineering-based roles. By forcing differing focuses on the teams in this setting, the students experience how the software engineering environment consists of compromises among different teams. Furthermore, unlike other research using software tools for the simulation [4][7], this approach is a human-enacted simulation [1][3][8].

This approach to software engineering education is split into two undergraduate courses. The first course brings all students to a common ground in the fundamentals of software

engineering with a large emphasis on software design, specifically object-oriented design. The second course introduces the idea of the software lifecycle. In addition, software architectures are discussed in conjunction with the concepts of software design from the original course. This second course culminates the training for the undergraduates with a semester-long simulation where the students form software engineering teams that have to accomplish roles that evolve as the semester progresses. This focus of this paper is on the second of the two software engineering courses, Software Engineering II. The subsequent section will discuss the collaboration training gained in this course. The next section will present the simulation portion of the first course offerings. The final section will discuss the results of the course offering, benefits and drawbacks, in addition to future plans.

2. Software Engineering II

The intention of Software Engineering II (COS346) is to give a real world experience to students interested in leading large software development projects through conceptualization, design, and development. The initial 4 weeks of the course opens with an introduction of current software architectures and a review of software design methodologies (a short review of Software Engineering I). Subsequently, members of the class split into the individual software development roles and design a software solution in support of some local information technology-based problem. At times, IT Corporate officers present and participate during the software development lifecycle. This course offers experience in the object-oriented software development lifecycle, design and programming using Rational Rose, latest software development technologies, project management using Microsoft Project and Configuration Management with ClearCase.

2.1. Software Engineering II Curriculum

The Software Engineering II curriculum tends to be a rigorous course. Students are expected to take part in a team and present deliverables each week. The first offering of this course was taught during a 16-week semester, where class (3 hours) is once a week. During the first 3 weeks, there are lectures on software architectures and the object-oriented software lifecycles. For the remainder of the course, class time is split between team presentations and inter-group collaboration. The team presentations consist of deliverable time-line updates by each of the team leads. The team leads may also designate members of their teams to present technical work accomplished during the week. Students are evaluated during the semester on the completeness, clarity, and timeliness of their deliverables. The inter-group collaboration discussions allow project-level directions and decisions to be accomplished. In fact, this inter-group collaboration is a major learning aspect of this course. The detailed 16-week syllabus is detailed in Table 2.1. The simulation portion of the course occurs from week 4 to the conclusion of the course and this time closely follows the software life-cycle found in the both the Rational Unified Process and the Analysis model of the Object Modeling Technique.

The course summarization as detailed in Table 2.1 can be somewhat misleading. The “Focus and Deliverables” column is not the only topics under consideration in those weeks. In fact, various teams may have different focuses for each week than what is presented in Table 2.1. The focuses and deliverables in Table 2.1 are the overall deadline for all of the teams. This section will define the teams and show how the roles of the teams and members evolve over the course.

Timeframe	Focus and Deliverables
Week 1: <i>Introduction and Review</i>	Review of Software Engineering and Object Technology (Software Engineering I)
Week 2-3: <i>Software Architecture and the Software Development LifeCycle</i>	Software Architecture (History and Different Types [11]), The Software Development Lifecycle, Introduction to the Rational Unified Process[2] [6] [9], Students Interview for desired positions
Week 4: <i>Project Initiation</i>	Functional responsibilities in the Life-cycle, Developmental Terminology, Org Charts, Interview recaps, Project procedures and milestones Gantt chart overview and demonstration (Microsoft Project)
Week 5: <i>Conceptualization</i>	Problem Statement Formation Proposed System Design, Requirements Elicitation introduction
Week 6-7: <i>Elicitation</i>	Requirements Elicitation, System Design Requirements Delivery, Finalize Problem Statement, Requirements, and System Design
Week 8: <i>Software Design</i>	Focus: Use Case Models, Scenarios, and Class Diagrams
Week 9: <i>Software Design</i>	Focus: Class Diagrams and Dynamic Models, User Interface Design
Week 10: <i>Software Design</i>	Focus: Deployment and Component Models, Wrap up pre-development designs
Week 11: <i>Development</i>	Coding/Spiral Development
Week 12: <i>Development</i>	Coding/Spiral Development
Week 13: <i>Development and Testing</i>	Coding/Spiral Development, Test Case Generation
Week 14: <i>Development and Testing</i>	Finish Coding, Start Testing
Week 15: <i>Deployment and Documentation</i>	Code Freeze, Complete Testing, Finish all Documentation
Week 16: <i>Executive Presentation</i>	Open Presentation to Faculty, Students, and Industry Stakeholders, Deliver Documentation

Table 2.1 Software Engineering II Weekly Schedule

2.2 Team Definitions

In the Software Engineering II curriculum, there are 4 teams, the analysis team, the software design team, the development team, and the database team. A chart of the teams is illustrated in Figure 2.1.

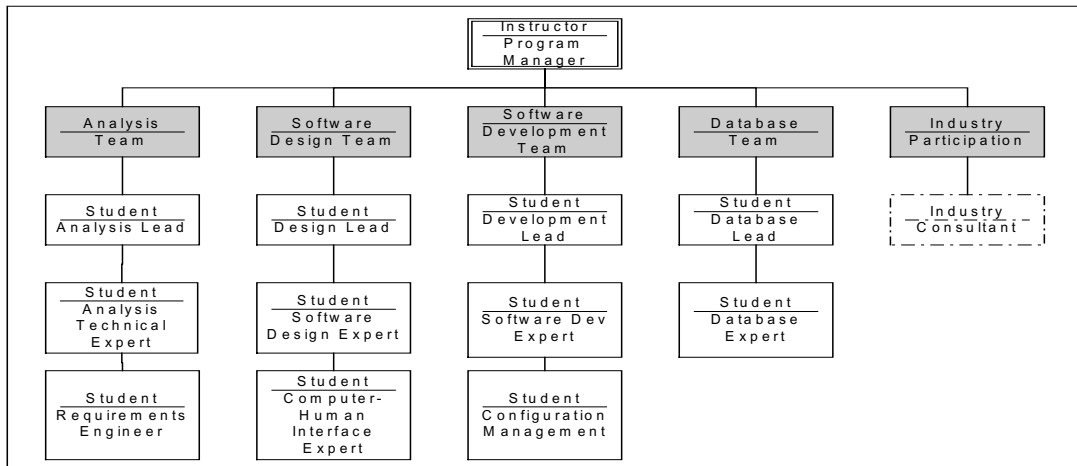


Figure 2.1. Organization in Software Engineering II

Each of the teams has one team leader that reports the progress of the team as a Gantt chart (Microsoft Project). However, each student has a particular specialized role for which he/she takes responsibility. The instructor for the course typically acts as the architect/program manager. In the event that there is not an industry problem, the instructor acts as the customer in order for the students to build their problem statement and requirements. The organization of the teams and the specialized team member roles is illustrated in Figure 2.1.

It is not necessary to have a one-to-one student-to-role mapping for the roles depicted in Figure 2.1. In fact, in the first offering of this class, two students represented each role. However, it is important that students be made responsible for the deliverables of a particular role, if possible. This allows them to be evaluated throughout the course on one aspect specifically. This evaluation can also occur on the final documentation delivered based on that role.

2.3 Evolving Roles

As the critical path of the software development lifecycle progresses, the activity of certain roles can increase and decrease as certain milestones are met. For example, the analysis role is the primary job (on the critical path) at the beginning of the project, but as the development process progresses the activity of this role decreases as the only responsibility is to assure that requirements are met. In an academic environment, the activity of all roles (students) must be maintained throughout the semester.

Critical Path	Analysis	Design	Developers	Database
Conceptual-ization	X	Org Chart	Configuration Management for Documents	Database Survey
Problem Analysis	X	Web Page	Technology Survey	Collecting Pre-existing Data
Requirements Elicitation	X	X	Set-up Development Environment and Configuration Management for Code	Set-up Database Environment
System Design	Assuring Requirements	X	Prototyping X	Set up Database Connection
Software and Database Design	Test Cases	X	X	X
Development	Test Scripts	X User Manual	X	X
Testing	X	Design Documents	X	X
Documentation	X	X	X	X

Table 2.2 Roles as the Critical Path progresses
(X's represent that the role is on the critical path)

Subsequently, this course promotes the evolution of roles as the semester progresses. One example of this evolution is in the analysis team. Once the critical path proceeds past analysis into project design, the analysis team becomes responsible for developing test cases and scripts. Another example is with software development team. Their roles cannot start until the domain is analyzed and the software is designed. However, they can take some responsibilities in surveying and presenting available pertinent technologies and prototyping them. The evolution for all roles with respect to the critical path is detailed in Table 2.2. An interesting benefit of evolving the roles is that it adds a process for “checks and balances”. The secondary role of one group can be used to preserve the correct decision-making of another group. For example, developers can do technology surveys and prototyping prior to system design and that in fact verifies the choice of solution architectures in software design.

The creation of test cases by the analysis team can predict development problems in the design team.

3. The First Offering of Software Engineering II

In the first offering of Software Engineering II, as aforementioned, the overall purpose of the course was to encourage the students to simulate a real-world development environment utilizing the concepts and methodologies learned in the first semester. The first step to the simulation portion of the class is to imagine a problem and a necessary but plausible solution. In the first offering of the class, many problems were investigated stemming from the shortfalls of the Georgetown University Information Services (UIS) and the Department of Computer Science. Future courses will incorporate problems from outside corporations. The class was forced to come to a consensus on an appropriate project with help from representatives from faculty in the Department of Computer Science and employees of the web development department of UIS. In this first phase, already students began to realize the rigorous nature of collaborations in the software development environment. A majority of the time in the preliminary meetings of the class was spent proposing and discussing a variety of problems and potential solutions.

The class decided on an approach that utilized Sun Microsystems' Java Servlets and Database connectivity technologies, which would also utilize the wave of popularity of the Internet and its applications. The simulation project was entitled GOTCHA (Georgetown Online Tool for Connecting, Hiring, and Accessing). The project worked as a small-scale example of an online phonebook for the Georgetown Computer Science Department. The first challenge of the students was to make adjustments to the problem statement and requirements to appropriately scope the problem to the semester timeframe. The target environment is a web interface (Java Servlets) connected to an Oracle database. Users of the system (students and faculty) input personal data, which is captured in the database. The system was designed to maintain this information, and present generic pages to users visiting the web site. In this way, faculty and staff would have automated means to maintain personal data such as contact information, resumes, office hours, etc without the burden of creating their own web pages. Instead, the system dynamically builds generic web pages based on information stored in the database. A screen shot of the GOTCHA user interface is captured in Figure 3.1.

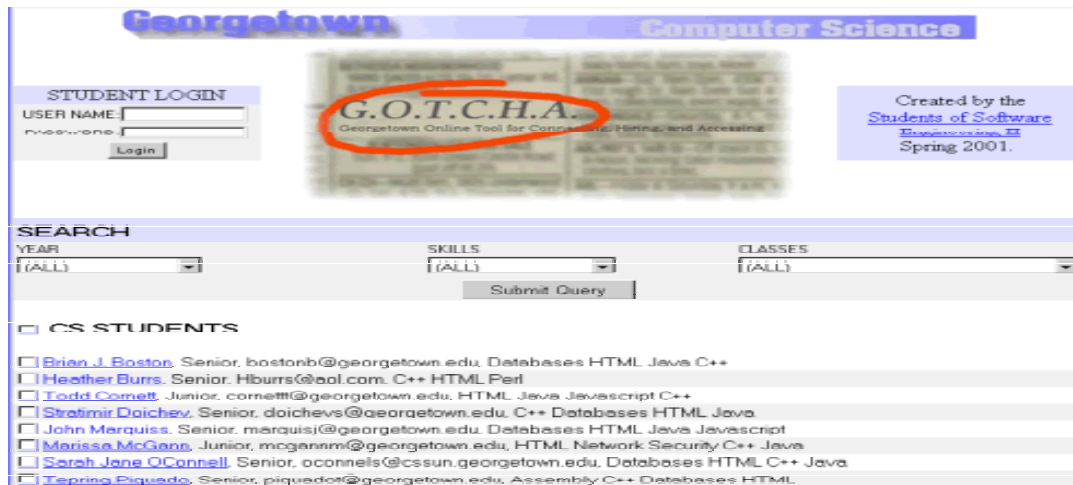


Figure 3.1 GOTCHA User Interface

4. Insights from the Course Offerings

This simulation approach to software engineering provided some helpful insight into undergraduate software engineering education. The students that took these courses were mostly graduating seniors with prior programming, analysis, and theory background. In the case of Georgetown University, this is the only application-based course offered to the students among a curriculum of theory courses. The main innovation in this course was the collaboration experience gained by students at two levels. Other group-based coursework as The Real World Lab at Georgia Institute of Technology [12] and The Studio at Carnegie Mellon University [13] emphasize “flat” collaboration (i.e. one big group with individual roles). This approach emphasizes team collaboration (between individuals) and group collaboration (between teams).

These courses would only be effective for classes of 20 or less. The team collaboration would not be effective for more than 5 individuals per team. More than 5 groups would make the group collaboration cumbersome, at least, in this academic setting. A potential change to the class would be to offer it for 4 credits, instead of 3 credits. The students tend to meet 2 times a week outside of class by their own choice. This time should be captured as an additional credit that could be represented as group collaboration/lab time. The class meeting times near the end of the semester always exceeded the 2.5 hours of class time, because of the magnitude of issues that had to be covered.

5. Conclusion

This practical approach to software engineering is a new approach for teaching undergraduates collaborative skills in addition to technical software engineering principles. The instructor of this course has eight years of consulting experience as a software project lead and technical lead. This experience led to the inspiration for the class. The students received the first offerings of both of these courses favorably. For the first offerings of Software Engineering I and II, they received evaluations of 4.5 and 4.75, respectively on a 5-point scale. The courses had 18 and 11 students, respectively. Considering both are new courses and the size of the computer science department at Georgetown University, these class sizes were above average.

Favorable comments of the course consisted of “I was able to use examples from this course in my job interviews.” “This course helped me understand the big picture in my internships.”, and “I was surprised that so much collaboration/argumentation occurred in the software development environment.”. In the future, the course project will be taken from industry. Software Engineering II will be offered again in the spring of 2003. We anticipate a software project in the area of scientific programming.

6. Acknowledgements

I would like to acknowledge the efforts of undergraduate students that participated in both classes and enhanced the findings of this research. These students are Todd Cornett (who also helped compile information for this paper), Tepring Piquado, Marissa McGann, Brian Boston, Nick Sklavounous, Stratimir Doichev, John Marquis, Sarah Jane O’Connor, and Heather Burrs. I also would like to acknowledge the support of the Rational Corporation for their generous grant of licenses of the Rational Rose Enterprise Edition 2000 software suite.

7. References

- [1] Boehm, B.W., et al., A Stakeholder Win-Win Approach to Software Engineering Education, in *Annals of Software Engineering*, O. Balci, Editor. 1998, Baltzer Science Publishers. p. 295-321.
- [2] Booch, G., Rumbaugh, J., and Jacobson, I. *The Unified Modeling Language Guide*. Addison Wesley, 1998
- [3] Dawson, R., Twenty Dirty Tricks to Train Software Engineers, in *Proceedings of the 22nd International Conference on Software Engineering*. 2000, ACM. p. 209-218.
- [4] Drappa, A. and J. Ludewig, Simulation in Software Engineering Training, in *Proceedings of the 22nd International Conference on Software Engineering*. 2000, ACM p. 199-208.
- [5] Garland, D. Software Architecture: A Roadmap. *The Future of Software Engineering*. New York, NY: ACM Press, (2000) 91-101
- [6] Krutchen, P. *The Rational Unified Process: An Introduction (2nd Ed.)*. Prentice Hall, 2000
- [7] Oh, E. and van der Hoek, A., Challenges in Using an Economic Cost Model for Software Engineering Simulation, in the n *Proceedings of the 3rd International Workshop on Economics-Driven Software Engineering Research*, Toronto, Canada, May 2001.
- [8] Port, D and Boehm, B. Using a Model Framework in Developing and Delivering a Family of Software Engineering Project Courses. *Proceedings of the 14th International Conference on Software Engineering Education and Training*, Charlotte, NC 2001
- [9] Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., Lorensen, W. *Object-Oriented Modeling And Design*. Prentice Hall, 1991
- [10] Shaw, M. Software Engineering Education: A Roadmap. *The Future of Software Engineering*. New York, NY: ACM Press, (2000) 371-380
- [11] Shaw, M. and Garland, D. *Software Architecture: Perspectives on an Emerging Discipline*. Prentice Hall, 1996
- [12] The Real World Lab, Georgia Institute of Technology, <http://www.cc.gatech.edu/classes/RWL/Web/>
- [13] The Studio, Carnegie Mellon University, <http://www2.cs.cmu.edu/afs/cs.cmu.edu/project/mse/www/courses/studio.html>