AC 2012-3347: TEACHING SOFTWARE ENGINEERING: AN ACTIVE LEARNING APPROACH

Dr. Walter W. Schilling Jr., Milwaukee School of Engineering

Walter Schilling is an Assistant Professor in the Software Engineering program at the Milwaukee School of Engineering in Milwaukee, Wis. He received his B.S.E.E. from Ohio Northern University and M.S.E.S. and Ph.D. from the University of Toledo. He worked for Ford Motor Company and Visteon as an embedded software engineer for several years prior to returning for doctoral work. He has spent time at NASA Glenn Research Center in Cleveland, Ohio, and consulted for multiple embedded systems companies in the Midwest. In addition to one U.S. Patent, Schilling has numerous publications in refereed international conferences and other journals. He received the Ohio Space Grant Consortium Doctoral Fellowship, and has received awards from the IEEE Southeastern Michigan and IEEE Toledo sections. He is a member of IEEE, IEEE Computer Society, and ASEE. At MSOE, he coordinates courses in software quality assurance, software verication, software engineering practices, real time systems, and operating systems, as well as teaching embedded systems software.

Dr. Mark J. Sebern, Milwaukee School of Engineering

Mark J. Sebern is a professor in the Electrical Engineering and Computer Science Department at the Milwaukee School of Engineering (MSOE), and founding Program Director for MSOE's undergraduate software engineering program. He has served as an ABET Program Evaluator for software engineering, computer engineering, and computer science, and is currently a member of the ABET Engineering Accreditation Commission.

Teaching Software Engineering: An Active Learning Approach

Abstract

Software Engineering is a core component of many computer engineering programs. In software engineering courses, students are taught to apply their programming and development skills to solve a larger scale problem. The resolution of this problem involves the development of an understanding of the problem from the client's perspective as well as an analysis of solution alternatives.

Unfortunately, in many cases, the software engineering course is offered late in the curriculum, typically at the senior level. This makes it difficult for students to apply the knowledge that they have learned effectively on capstone and other academic projects. Students often comment that it would have been "nice to know this" before making the wrong decisions on their capstone projects. Thus, to be successful, components of software engineering need to be taught earlier in the undergraduate curriculum. This shifting to an earlier level, however, poses pedagogical issues.

This paper describes the metamorphosis of an undergraduate software engineering course from a senior level course to a sophomore level course. In this course, students are taught to use software engineering tools and practices in pursuit of a solution to a software based embedded systems problem. Students actively work together in teams while theoretical software engineering concepts are delivered using "Just in Time" instruction.

In addition to providing an overview of the course material and exercises, this article will discuss the changes made to the course in each of the previous 4 offerings. Changes were based upon student comments and other feedback. An analysis of student performance will also be provided.

Introduction

The Milwaukee School of Engineering offers two computer related engineering fields, a Bachelor's of Science in Software Engineering degree and a Bachelor's of Science degree in Computer Engineering. For many years, seniors within the Computer Engineering field took a required course, *CS-489 Software Engineering Design*, as is shown in Figure 1. The course was designed as a project based course providing a survey of software engineering methods as well as introducing a design process for their capstone projects. Details of this course are provided in Sebern[1] and Welch[2].

Freshman Year		Fall	Winter	Spring
GE-110	Introduction to Engineering Concepts	2-2-3		
CS-1010	Computer Programming		2-2-3	
CS-1020	Software Design I			2-2-3
Sophomore Year				
CS-1030	Software Design II	2-2-3		
CS-2851	Data Structures		2-2-3	
CS-280	Embedded Systems Software			3-2-4
EE-210	Electronic Devices and Computer Interfacing			3-3-4
Junior Year				
CS-321	Computer Graphics	3-3-4		
CS-381	Engineering Systems Analysis With Numerical Methods	3-2-4		
CS-384	Design of Operating Systems		3-2-4	
CS-393	Computer Architecture I		3-2-4	
CS-391	Embedded Computer System Design			3-3-4
Senior Year				
CS-489	Software Engineering Design	3-3-4		
CS-495	Data Communications and Networking	3-3-4		
CS-400	Senior Design Project I		2-2-3	
CS-401	Senior Design Project II			2-2-3

Figure 1 MSOE Computer Engineering Computer Programming Related Curriculum Courses (Version 2.12)

Based on senior exit interviews (samples of which are given in Figure 2) and other course feedback, there was a strong consensus that the placement of the software engineering course was too late in the curriculum. As part of a curriculum overhaul, a decision was made to convert the existing software engineering course into a sophomore level course[2]. This resulted in the Computer Programming related sequence shown in Figure 3. In addition to moving the course sooner in the curriculum, the course also received a credit reduction, reducing both the lecture and lab contact hours by 33%. A second change that should be noted is that the number of contact hours associated with the introductory programming courses was also reduced 25% from 8 contact hours of lecture and 8 contact hours of lab to 6 contact hours of lecture and 6 contact hours of lab. This change was accomplished by combining CS-1010, CS-1020, and CS-1030 into two courses.

- "Should be introduced to us earlier and more often than just Software Engineering Design."
- This was only given to us in Software Engineering Design where we had to cram too much information in.
- The only real attempt to apply the principles of team process and project management came in Software design, which was much too late to be of much help.
- The project in the SE class for CE's was what I would consider the first real team project that I'd participated in. What I learned about teamwork and team roles would have been extremely helpful earlier on. As I mentioned before, this class needs to move to earlier in the curriculum.
- More Software Engineering in Computer Engineering: It seemed kind of strange that the only class that covered software engineering processes was only taught in senior year. I think a lot of the software design knowledge would have been more beneficial at an earlier state. A lot of the code that everyone wrote was hacked out instead of following a proper design procedure.

Figure 2 Sample student comments from exit interviews about CS-489 course.

		Fall	Winter	Spring
Freshman Year				
SE-1010	Software Development I	2-2-3		
SE-1020	Software Development II		2-2-3	
CS-2851	Data Structures			2-2-3
Sophomore Year				
CE-2800	Embedded Systems I	3-3-4		
CE-2810	Embedded Systems II		2-2-3	
SE-2890	Software Engineering Practices			2-2-3
Junior Year				
CS-3212	Computer Graphics	2-3-3		
CS-3841	Design of Operating Systems		3-2-4	
CE-3910	Embedded Systems III			3-2-4
Senior Year				
CE-4000	Senior Design Project I	2-2-3		
CE-4920	Embedded Systems IV	2-2-3		
CE-4010	Senior Design Project II		2-2-3	
CE-4950	Networking I		2-2-3	
CE-4020	Senior Design Project III			2-2-3
CE-4960	Networking II			2-2-3

Figure 3 MSOE Computer Engineering Computer Programming Related Curriculum Courses (Version 3.0)

The New Course Initial Offering

The initial inception for the new *Software Engineering Practices* course was to offer students a scaled down version of the senior level course, removing some topical content, such as formal methods and client interviews, but otherwise, retaining the initial flavor of the senior level course. For the initial course offering, the 13 course outcomes listed in Figure 4 were defined for the course, and the syllabus of Table 1 was used for lecture and lab content.

Catalog Description:
This course presents an introduction to the team-based cyclical development of software for non-SE majors. Computer-aided software engineering (CASE) tools are used to support the development process, which is built around the objected-oriented (OO) paradigm and will reinforce understanding of the Unified Modeling Language (UML). Students participate in a team project to analyze, design, implement and test a complete software system.

Course Outcomes:

1. describe the software engineering life cycle
2. analyze and generate simple use cases
3. apply object-oriented analysis techniques to small projects and represent them using UML
4. apply object-oriented design techniques to small projects and represent them using UML
5. describe the purpose of and apply basic design patterns
6. analyze and document software system requirements using OOA techniques
7. design and implement software systems using OOD techniques
8. generate clear, consistent, and reasonably complete documentation of a software system
9. be able to use computer-aided software engineering (CASE) tools
10. develop basic software test plans and reports
11. work effectively as part of a team
12. apply simple quality project monitoring techniques
13. describe the purpose and goals of the SEI Capability Maturity Model

Figure 4 Initial SE2890 Course Outcomes

Table 1 SE2890 Initial Course Syllabus

Week	Day 1 Lecture	Day 2 Lecture	Lab	Assignment(s)
1	Introduction The Software Crisis	Use Cases	Java Refresher and Data Collection	Team Member Resume
2	Requirements Domain Models	System Behavior Sequence Diagrams Contracts	OO CASE Tool and Effort Estimation	Lab 1 Report
3	Interaction Diagrams	Assigning Responsibilities	Project, Cycle 1: Analysis	Lab 2 Report
4	Designing Solutions	Design Class Diagrams	Project, Cycle 1: Design	Cycle 1 Analysis Document
5	Architectural Issues	Implementation Techniques	Implementation Project, Cycle 1: Design cont.	Status Memo
6	Midterm Exam	Code Reviews Checklists Code Review Procedure Sample Checklist	CASE Tool Demo: Project, Cycle 1: Implementation & Test	Cycle 1 Design Document
7	Project: Code Review	Testing Configuration Management	Project, Cycle 1: Implementation & Test cont.	Status Memo
8	Test Case Exercise	Generalization	Project, Cycle 2: Analysis & Design	Cycle 1 Final Report Peer Evaluation Role Summary
9	Additional Design Patterns	Software Metrics Estimation	Project, Cycle 2: Analysis & Design cont.	Cycle 2 Analysis & Design Report
10	SEI CMMI Web Site	Review for Final Exam	Project, Cycle 2: Implementation & Test	Final Project Report Peer Evaluation Role Summary

With the initial offering of the course (Spring 2008), there were many problems noted, ranging from course content to prerequisite material.

One of the most surprising observations was the difficulty students had with fundamental object-oriented programming. During their freshman year, they had taken three courses using Java (SE1010, SE1020, and CS2851). However, it had been nearly a year since the students had used any programming language other than assembly or C. Thus, there were significant retention issues with the Java programming language, making it very difficult for students to be successful. Many students required additional tutoring in the fundamentals of the Java programming language, and even then, their success was marginal.

Students also had great difficulty becoming engaged in the lab project. While the project itself did not represent an unrealistic software engineering task, it also was not a project that engaged the students and made them really want to learn the material, especially given that they had been given a tool to use with the first two labs which essentially met the requirements for the final project. Because the lab project was built in two cycles, it was also very difficult to provide the students with meaningful feedback in a timely enough fashion for them to integrate that feedback into the development of the next cycle.

Exit surveys from the course also indicated problems with the revised course. For each outcome, students were asked on a Likert scale to answer whether the student had obtained an understanding in the given outcome and whether the course material had helped them to obtain this understanding. Overall, students felt that they were weak in outcomes 5, 9, 12, and 13.

Furthermore, the students felt that the course as structured only helped to learn the material related to outcome #2. Written comments from students also indicated problems from the course as well. Many indicated they did not feel there was a flow to the course, and many strongly felt that the course was very rushed. Other students also felt that they were at a significant disadvantage in that pre-requisite courses had not covered the material they were expected to know at the beginning of the course.

The First Revision of the new course

Based on student comments and outcome assessments from the first course offering, a significant redesign of the course was approached. Each outcome was carefully reviewed and compared against the needs of computer engineering students as well as checked for overlap with existing courses. From this, the initial listing of 13 course outcomes was reduced to 11 course outcomes, as is shown in Figure 5. These outcomes reflected an increased emphasis on the practices necessary for software engineering and a slight de-emphasis on the specific object-oriented analysis and object-oriented design techniques from the initial offering. An increased emphasis on the verification of software was added, as well as outcomes related to effective communications. Based on feedback from the final offering of CS489, it was also felt that a strong emphasis in the area of requirements review and development was necessary.

Catalog Description:
This course provides an introduction to the discipline of software engineering for Non-majors. Students will be exposed to the practices employed in determining requirements for the software which is to be developed. From the requirements specification, problem domain analysis will lead to a high level design. After review, the high level design will be used to create detailed designs and implement the software on a desktop machine. These activities will be reinforced through a team project and culminating with group oral presentations.

Course Outcomes:

1. Recognize the risks of software failure and appreciate the importance of a disciplined software development approach.
2. Compare and contrast distinct models for software development.
3. Employ rudimentary configuration management tools and processes across a software development project
4. Verify through the practice of review that specified requirements are accurate, unambiguous, complete and consistent
5. Apply UML modeling tools to represent all phases of a software engineering project
6. Conduct efficient and effective software reviews, and measure the effectiveness of those reviews
7. Perform rudimentary software testing using both manual and automated mechanisms
8. Demonstrate independent learning to accomplish tasks for which all of the details may not have been taught in previous courses.
9. Work effectively in a team environment on a short-term software development project
10. Communicate design and implementation judgment to others through a team-based oral presentation
11. Demonstrate effective written and oral communications skills

Figure 5 Modified course outcomes and catalog description for SE2890.

The course syllabus was also redesigned so that material was presented in a slightly less intense format as well as in a linear fashion. The revised syllabus is shown in Table 2. Key supporting concepts (such as reviews and configuration management) were moved earlier in the course in order that students would be able to use this material throughout the course. And, while certain topics were kept in the syllabus, they were cast in a more applicable fashion. For example, the initial course covered 8 design patterns in a single lecture while the new course focused on two design patterns in the same amount of time. This additional emphasis allowed more time for the students to see the development and application of the design patterns. Project tracking was left last in the course in order that the actual data collected by the students could be presented back to the students as a retrospective on their projects. The redesigned syllabus also incorporated Just-

Table 2 SE2890 Modified course curriculum.

Week	Lecture	Topic
1	1	Introduction to Software Failure
1	2	Software Development Processes
2	1	Requirements and Use Cases
2	2	Requirements and Use Cases
3	1	Introduction to Software reviews
3	2	Configuration Management
4	1	Object Domain Analysis
4	2	Object Domain Analysis
5	1	Defining Object Behavior
5	2	Defining Object Behavior
6	1	Design and Design Patterns
6	2	Midterm Exam
7	1	Detailed Design
7	2	Implementing State Charts in Source Code
8	1	Software Testing
8	2	Software Testing
9	1	Code Reviews
9	2	Code Reviews
10	1	Project Tracking and Analysis
10	2	Course Evaluation and Final Review

In-Time teaching, where students actively learned during lecture the materials that would be needed for the lab project.

Team selection was also modified. In the previous offering, students had written a resume of their experience and submitted it to the instructor for team assignments. This was problematic in that it did not allow the instructor to take into account student schedules and other factors. Thus, for the second offering, the CATME[3] system was selected to manage team formation and handle team assessment.

The largest change between the first and second offering was in the area of the lab. For this course, the lab sequence was completely rewritten. Whereas the first offering attempted to do two cycles over the course of 8 weeks (yielding approximately 6 lab hours for the second cycle), the lab was recast to be a single cycle of a larger project. This meant that a greater amount of time dedicated to each cycle, and therefore, more time to do a quality deliverable. This modified lab sequence is shown in Table 3.

Table 3 SE2890 Modified Lab Sequence

Week	**Topic**
1	Java Programming Review
2	UML Case Tools and Class Design
3	Requirements analysis and SRS Review
4	Object Domain Analysis Development
5	Defining Object Behavior
6	High Level Design
7	Detailed Design
8	Implementation
9	Testing
10	Project Oral Presentations / RC Car Olympics

A second change was that requiring students to use the GForge project management system. GForge was chosen for the students to use because this was the required tool for students to use during their senior design capstone course, and it was felt that teaching them how to use the tool at this level would help them when they became seniors.

A third significant change between the first offering of the course and the second offering involved a shift in domain for the lab project. For the lab project, students were responsible for constructing software which would control an RC car, shown in Figure 6, via a wireless USB interface. Students were provided with an interface file and device driver, but were responsible for all other aspects of the design and implementation of the vehicular control system. This involved the construction of a GUI, construction of multiple state machines to control vehicular operation, and integrating the device driver into their project.

A fourth change involved how the lab time was managed. In the previous offering, the instructor returned assignments in a routine fashion after grading had been completed. This unfortunately did not give the required immediate feedback to the team. To avoid this problem, the instructor of the class held a design review the submitted artifact with each team during a portion of the lab session. In this review, the instructor would go through the submitted artifact with the team and provide oral feedback and critique. Traditionally grading against the grading rubric then occurred at a later time. However, by doing it in this manner, teams could immediately fix glaring problems with submitted artifacts.

Figure 6 RC Car used by the students for lab.

The last major change between the first and second offering of the course was an increased emphasis on oral communications. This was driven by two factors. In the previous course, one instructor had used an oral final exam to assess the communication skills of individual students. While for this particular instructor it worked effectively, it was generally not condoned by the students or campus administration. Secondly, feedback from the capstone course indicated that students had significant problems giving technical presentations. Thus, the oral presentation was added as a conclusion to the lab project. This was approximately ½ of the time of the last lab session. The other half of the lab session was devoted to the RC Car Olympics. In the RC Car Olympics, teams competed against each other attempting to complete various tasks with their

developed system. Three main events were held, the RC Car Slalom race, RC Car Parallel parking, and the 10 foot drive. While the competition was mainly for the fun of the students, in some cases, it was possible to show different levels of performance based upon design decisions made earlier in the course. For example, one team which used a slow polling approach to steering had significant difficulty completing the slalom event, while a second team which did an exemplary job managing their distance measurement system was able to measure the distance traveled by their car to within ½ inch.

Student perspective from the revised course

While the revised course (Spring 2009) received better evaluations than the initial version, there still were some areas of concern. Given the new lab project, students needed to have a few additional fundamental skills in the Java programming language that were not taught in earlier courses. Another common thread in student comments was that, given the nature of the lab, it was very difficult to parallelize the design and implementation, as all of the team members really needed access to the RC car to test their individual contributions. The quality of the RC car also was a problem, as all of the RC cars used the same wireless communications channel. Thus when two teams were working in lab side by side, whichever team activated their car first controlled both cars. (In deference to the RC car manufacturer, the RC car was never intended to be used as anything more than a novelty item.)

Overall, the class felt that waiting until week #8 to commence implementation was too late, and only providing a single week for implementation was also problematic. A certain set of students felt that the first two labs were not necessary, because aside from some limited metrics analysis they were reviewing fundamental programming skills. However, other students vehemently felt that these were very important, as the on track CE students had not taken any Java courses since the previous year. Students still commented that the course was too rushed to be successful, and because of this, it was very hard to understand the material.

Quantitative assessment data from the final course surveys is provided in Table 4. Overall, the areas of greatest concern were those which warranted scores less than 3.0, namely “I felt that the workload for this course was appropriate given the course credit” and “I would recommend this course to CE student’s even if it was not a required course in the curriculum.” While “I learned a lot from this lab” rated below 3 for both of the two initial labs, this score was downplayed slightly based upon the written feedback from the students for the reasons stated previously. One other side note that should be included is there was a great amount of discontent over a schedule change made to the course the week before it was offered, requiring approximately 50% of students to change lecture times for the course. This may have negatively biased some of the responses.

Table 2 Quantitative survey data from the revised offerings of SE2890.

Data collected using a 5 point Likert scale, with 5 representing "Strongly Agreeing" with the statement and 1 represent "Strongly disagreeing" with the statement. The 4/5 percent represents the percentage of respondents who agreed with the statement. Cells marked in red received evaluations of less than a 3 on a 5 point scale.

		2009				2010				2011				Overall		
		Average	Median	Stdev	4/5 percent	Average	Median	Stdev	4/5 percent	Average	Median	Stdev	4/5 percent	Average	Median	4/5 Percentage
	Sample Size	18				19				17						
Basic Math Operations	This lab was an excellent tool for teaching the material I needed at the time.	3.28	3.50	0.89	50%	3.79	4.00	0.71	78%	3.65	4.00	0.86	71%	3.66	3.94	71%
	I learned a lot from this lab.	2.67	2.00	0.97	28%	3.21	3.00	0.98	39%	3.29	3.00	0.85	47%	3.18	2.87	41%
	This lab should be continued for future students	3.17	3.00	0.86	33%	3.63	4.00	0.83	67%	4.12	4.00	0.33	100%	3.79	3.87	78%
A Banking Test System	This lab was an excellent tool for teaching the material I needed at the time.	3.17	3.00	0.99	39%	3.63	4.00	0.83	72%	3.65	4.00	1.00	71%	3.58	3.87	67%
	I learned a lot from this lab.	2.89	3.00	0.96	22%	3.26	3.00	0.99	50%	3.53	4.00	0.87	59%	3.34	3.46	51%
	This lab should be continued for future students	3.22	3.00	0.88	39%	3.37	3.00	0.90	50%	3.88	4.00	0.70	82%	3.58	3.46	63%
The Lab Project	This lab was an excellent tool fo teaching the material I needed at the time.	3.00	3.00	1.08	39%	4.16	4.00	0.76	94%	3.88	4.00	1.05	76%	3.89	3.87	79%
	I learned a lot from this lab.	3.22	3.00	0.94	39%	3.89	4.00	0.74	83%	4.18	4.00	0.53	94%	3.94	3.87	83%
	I felt that Gforge greatly helped to complete this project.	3.39	4.00	1.33	67%	3.32	3.00	1.20	50%	2.47	3.00	1.12	18%	2.94	3.13	37%
	I felt that the workload for this lab was appropriate given the course credit.	2.83	3.00	0.99	28%	3.53	4.00	0.90	61%	3.53	4.00	0.51	53%	3.44	3.87	53%
	I enjoyed working in groups and this should be continued.	4.00	4.00	0.69	89%	4.21	4.00	0.79	83%	3.94	4.00	0.83	76%	4.06	4.00	81%
	This lab sequence should be continued for future students	3.28	3.00	0.89	44%	3.74	4.00	0.81	83%	3.94	4.00	0.83	76%	3.77	3.87	75%
The Course in General	At at least one point in this course, I found the course to be mentally challenging	3.18	0.00	0.00	0%	3.78	4.00	1.06	72%	3.82	4.00	0.73	76%	3.72	3.49	65%
	I learned from the course, and I believe I will do better on my senior design project because of this course.	3.22	3.00	0.88	44%	3.89	4.00	0.83	72%	3.94	4.00	0.56	82%	3.83	3.87	73%
	This course would be better sited as a senior level course	2.11	2.00	0.90	11%											
	I felt that there was too much time spent on software process instead of value adding activities.	3.39	3.50	0.98	50%	3.72	4.00	0.75	56%	3.47	3.00	1.01	35%	3.57	3.48	46%
	I felt that this course was rushed in the amount of content included versus the amount of lecture time provided.	3.33	3.00	1.19	39%	3.06	3.00	1.06	33%	2.94	3.00	0.97	24%	3.04	3.00	30%
	I would recommend this course to CE students even if it was not a requuired course in the curriculum	2.50	2.50	0.99	17%	2.78	3.00	1.00	22%	3.24	4.00	1.09	59%	2.95	3.39	38%

Course Revisions for the 2010 Offering

Based on course comments, a few minor changes to the course content were made for the second running of the revised course (Spring 2010), and these changes resulted in improved assessment scores, also shown in Table 4. In terms of the lab, small "proof of concept" implementation steps were started earlier in the sequence. The lab time allocated to design was reduced by one lab period, allowing an additional lab period for implementation.

One problem that again showed up was the difficulty students had learning new concepts that were necessary to complete the project. In specific, students commented that they had great trouble teaching themselves the concept of threading in Java. The students also had further quality problems with the vehicle.

Course Revisions for the 2011 Offering

For the Spring 2011 offering, the course received more significant changes. Due to the discontinuation of the RC car, the lab project had to be completely changed. This allowed a move was made to the LeJOS platform and Lego Mindstorm robots. This platform offered a tested environment as well as the ability to write and execute Java code on the NXT controller.

The risk in making this change is that the scope of the lab project grew slightly, as the students now needed to write software for both the robot and the PC whereas previously only software for the PC needed to be developed. In making this change, the lecture content was revised slightly to include brief coverage of Java threading, as this skill was even more important with the Mindstorm robots than it had been with the RC car. This modified lecture and lab sequence is shown in Tables 5 and 6.

Table 5 SE2890 Modified course curriculum for the 2011 offering.

Week	Lecture	Topic
1	1	Introduction to Software Failure
1	2	Software Development Processes
2	1	Requirements and Use Cases
2	2	Requirements and Use Cases
3	1	Introduction to Software reviews
3	2	Configuration Management
4	1	Object Domain Analysis
4	2	Object Domain Analysis
5	1	Defining Object Behavior
5	2	Defining Object Behavior
6	1	Midterm Exam
6	2	Design and Design Patterns
7	1	Detailed Design and Java Threading
7	2	Implementing State Charts in Source Code
8	1	Code Reviews
8	2	Software Testing
9	1	Software Testing
9	2	Project Tracking and Analysis
10	1	Applications to Embedded Systems
10	2	Course Evaluation and Final Review

Table 6 Modified Lab sequence for the 2011 course offering.

Week	**Topic**
1	Java Programming Review
2	UML Case Tools and Class Design
3	Requirements Specification and Use Case Construction
4	Use Case and Requirements Peer Review
5	Object Domain Analysis Development
6	Project Design
7	Implementation – Week #1
8	Code Review and Implementation – Week #2
9	Testing
10	Project Oral Presentations and Robot Olympics

Student Perspectives for the 2011 Offering

Overall, there was a slight decrease in some assessment scores between the 2010 and the 2011 course offering. This is most likely attributable to technical problems with the LeJOS system and the newly deployed Windows 7 64 bit images on the student's laptops. In essence, in order to make the robots work on the students' machine, there were a lot of very detailed installation instructions that needed to be followed exactly or problems would occur. Students also had problems with the instructor provided proxy code. Under certain circumstances, it would simply lock up, requiring the students to reboot. Unfortunately, this could not be duplicated on the instructor's machine. (Note: Subsequent to the completion of the class, a new version of the LeJOS environment was released which fixed many of the issues. This new release, however, was received during final exam week and could not be applied by the students to their projects.)

- Good introduction to planning software and breaking up programming among people.
- Fun to be able to work on a new platform. Also my first massively multi-threaded systems development (was fun)
- I have already applied the processes from class to my own personal process.
- Learned a lot about SE process and plan on applying it to future projects
- I liked slowly building up towards implementation and the design process.
- I liked learning the process.
- The overall project was good for exercising the skills learned.
- Good project I enjoyed it.
- (What did you like best about the class) the Mindstorm project.
- Learning process in an easy, fun way.
- Students should be able to choose their own groups.
- Bad group and didn't like svn.

Figure 7 Sample student comments from the 2011 course survey. Note that the response rate for written comments was 100%.

In general, students were really engaged in the lab project, more so than in previous years. A few students actually became slightly overly engaged to the detriment of other courses. There also was a significant improvement in the percent of students who would recommend the course to other students even if it was not a required course. The one area of concern the student surveys would post was related to the configuration management system, as the majority of the students did not feel this tool helped them to complete their project. Part of this is attributable to a few major system outages at very inopportune times through the quarter. A second reason this may have developed is a move by the IT department to place the GForge server behind a campus firewall, making it very difficult for off campus students to access the configuration management server.

For the 2011 offering, there also were fewer students having problems with the first two individual labs. This can be traced to a change in the introductory curriculum made in the freshman year. Based on poor assessment data from follow on classes, the three course introductory programming course received a 50% increase in lecture contact time, going from 2 to 3 lecture hours per week. This additional contact time vastly improved student's retention and understanding of the core programming material.

The 2011 offering marked the first year that the instructor received significant complaints about the assigned teams. While the number of negative comments was small, the complaints were generally traceable to a few students who were technically unprepared for the class and did not put forth effort to be successful. Unfortunately, this manifested itself negatively in the team experience for two different teams.

Observations and Conclusions

Having taught this course multiple times, there are many observations that can be made about teaching a design course at this level.

First off, for a course such as this to be successful at the sophomore level, the student comprehension and outcome obtainment for the prerequisite courses is essential. A software engineering course focuses on the approaches necessary to solve a problem, and as such, it cannot be successful if students do not have the basic fundamental skills necessary to solve the given problems. In this course, the first two labs are written to teach the students a little bit about time estimation and tracking. But, more importantly, they also serve as prerequisite assessments, for the students that have difficulty successfully completing the first two labs will most likely have significant problems with the team exercise. By having those labs up front, it serves as a mechanism for alerting the instructor to the students who will need more significant guidance later on in the course. Related to this notion, it is vital that the prerequisite course requirements be both correct and rigorously enforced.

Second, while there are distinct advantages to moving a design course forward in the curriculum, when implementing the course, it is important to remember that there may be significant skills that the students will either need to be taught based upon the assigned projects or the projects may need to be tailored to avoid these knowledge deficiencies. In this particular class, threading is a prime example of this problem. The initial lab sequence was constructed without threading in mind, as threading was not traditionally taught until the operating systems course in the Junior year. However, because threading is ubiquitous with modern software development in Java, either the students needed to learn the material on their own (which was tried in the 2009 and 2010 course offerings) or the students need to have brief coverage of the material within the course (as was done in the 2011 offering). Doing too much of this may disrupt the ability of the class to meet the core outcomes, while doing too little may inhibit student's success.

Third, it is also very risky to try and teach to sophomore the same amount of material and at the same depth as is taught to the seniors. Because of their greater experience with learning, senior students do not need the level of repetition and rigor needed to be successful with sophomore students. To learn the same amount of material requires more contact and effort on the part of the instructor at a sophomore level than it will at a senior level. This especially plays out in this course in the reduction in lab contact time from 3 to 2 hours. With a lab size of 20 or more students, 2 hours provides the instructor with approximately 5 minutes per week to work with each student during the introductory labs and approximately 25 minutes to work with each team per week once the team project commences. This reduction is clearly problematic. The lecture time reduction also poses issues, for if a given lecture is canceled or otherwise unable to be held, it is very challenging to get back on schedule with only 2 lecture hours per week. This issue definitely contributes to the students "whirlwind" comments about the course.

Exit Interview Comments:

- "Team-building" classes were frustrating and not very helpful. Another class like SE2890 focused more on teams than process would have been better.
- Servant Leadership classes were a bit contrived, although they were helpful to some extent with team process. SE2890 is really the only other team process and process management interaction I can remember having (besides Senior Design, of course)
- More emphasis on software design models and development processes would have helped with job seeking.
- SE-2890 was by far the most helpful team-oriented class thus far.
- Software engineering practices was a good start.

Figure 8 Sample senior exit survey comments regarding SE2890.

Fourth, in offering the course to sophomores, it is important that the project be engaging, even if the project starts to be more contrived in order to be engaging. In the case of this specific course, the first set of students was not interested in the line of code counting tool because to them it was an unexciting experience. However, subsequent classes were more engaged by the prospect of doing a software engineering project with an RC Car or Lego Mindstorm robots, and this led to greater achievement in the course itself. This means that more care must be taken when selecting the project in order to ensure that it meets both the scope and technical skills of the students.

Fifth, when dealing with teams at the sophomore level, the instructor must be significantly more proactive than when dealing with teams at the senior level. This poses an additional load on the instructor, for while earlier courses are tended to be viewed as easier to teach because the material is not as complex, teaching design at an earlier time actually requires more involvement than if it is deferred, and university loading calculations do not tend to reflect this difference.

Last, with a design course like this, it is important not only to measure the effects of the course at completion time, but also later on in the curriculum, as sophomore students may not have the ability to accurately assess their own learning needs. Figure 8 includes selected exit interview comments related to SE2890 made by exiting seniors during the senior debriefing. These comments show that at least for a set of students, they feel the course was significantly beneficial.

Bibliography

[1] Sebern. "Evolving an Undergraduate Software Engineering Course." ASEE Annual Conference, Milwaukee, WI, 1997.

[2] Welch "Teaching a service course in software engineering," Frontiers In Education Conference - Global Engineering: Knowledge Without Borders, Opportunities Without Passports, 2007. FIE '07. 37th Annual , vol., no., pp.F4B-6-F4B-11, 10-13 Oct. 2007
doi: 10.1109/FIE.2007.4418062

[3] Layton, R.A., M.L. Loughry, and M.W. Ohland, "Design and Validation of a Web-Based System for Assigning Members to Teams Using Instructor-Specified Criteria," in press, Advances in Engineering Education.