Teaching Software Development Skills Early in the Curriculum Through Software Engineering

C. W. Liew
Department of Computer Science
Lafayette College
Easton, PA 18042
liew@cs.lafayette.edu

ABSTRACT
Software engineering is commonly taught as an upper level course in many computer science departments. This paper describes our experience in teaching software engineering to second year students in computer science. There are many software development skills that can be taught in a software engineering course that are useful in other courses. The development of this course has greatly influenced the types of projects and assignments that are required in the upper level courses. The upper level courses have been able to (1) develop more challenging projects and (2) reduce or eliminate the time spent on teaching development skills.

Categories and Subject Descriptors
D.2.0 [Software Engineering]: Curriculum Issues

General Terms
Design

Keywords
software development skills, software engineering curriculum

1. INTRODUCTION
Software engineering is commonly taught as an upper level course in many computer science departments (e.g., [7]). However, many of the software development skills that are acquired could be used in other upper level courses, e.g., working in teams on Operating Systems and Computer Networks projects. Some of the desired skills include the ability to:

- develop software in teams,
- develop requirements and specifications,
- design and implement medium sized programs,
- use “standard” software maintenance tools such as make[8], debuggers (gdb) and source code control systems, e.g., CVS[9] or RCS[24],
- understand and use application program interfaces (APIs) for new software packages and languages,
- design and implement user interfaces (UI) for programs,
- effectively use the data structures covered in a standard CS 2 course,
- systematically test software units and modules,

At Lafayette college, a software engineering course that teaches these skills has been successfully taught to computer science and computer engineering students in the spring of their second year and sometimes in their first year. One desired result is that group projects requiring the use of software development tools like makefiles and CVS have been successfully completed in several upper level courses without any instruction in the use of those tools. In addition, the students have successfully designed and implemented GUIs in many of these courses. These skills are useful in both upper level courses and in industrial internships. The students also feel that the skills make it easier for them to obtain summer positions.

There is therefore, much that can be gained if software engineering is taught early in the curriculum. This paper describes the software engineering course at Lafayette college and our experience with having software engineering as one of the foundational pieces in the curriculum. McCauley and Jackson [16] have previously described their positive experiences with teaching software engineering early in the curriculum. Their approach used documentation and design standards as a framework for teaching software engineering principles and techniques. This approach is one way of dealing with the integration of software engineering into the curriculum [13]. Tewari [23] among others have alternatively focused on software reuse in their software engineering courses. Humphrey’s Personal Software Process [12, 11] focuses on the improvement of software engineering skills whereas we focus on the initial phase of learning the skills. Our course focuses on teaching the software development skills that are useful in other upper level computer science and computer engineering courses. This approach has allowed us to revise our curriculum using the software engineering course as a common pre-requisite to the upper level courses.
2. BACKGROUND AND PREREQUISITES

Lafayette College is a four year liberal arts college with an engineering school and it offers both a computer science and an electrical and computer engineering degree. The students taking computer science courses are mostly computer science and electrical and computer engineering majors. In particular, electrical and computer engineering majors are required to take CS 102 (Introduction to Computer Science), CS 103 (Data Structures), and CS 205 (Software Engineering). The computer science majors take software engineering in their second and sometimes third year (if they entered the major in the second year) while the electrical and computer engineering majors take the course in their second and third and sometimes fourth year.

CS 102 (CS I) and CS 103 (CS II) are taught in Java on Windows platforms while all subsequent computer science courses including CS 205 (Software Engineering) are taught using Windows XP workstations as front ends to Linux (currently Red Hat Linux 7.3) machines where the program development and maintenance is done.

3. COURSE STRUCTURE AND MATERIALS

The software engineering course at Lafayette College covers all the topics listed in the introduction (see section 1) along with a few others that are specific to our particular program. The additional topics that are covered in the course include:

- **C++ language**: This is a specific requirement of the electrical and computer engineering majors. CS 1 and CS 2 are both taught in Java. Since the computer science department also wanted to have the students be familiar with the language, CS 205 was the obvious place for them to learn the material. The material is covered in twenty small assignments that focus primarily on teaching the students (1) the syntactic differences between C++ and Java and (2) the syntax and use of pointers. The assignments describe programs in pseudo code that the students have to then implement in C++, i.e., they do not need to design the program. The basic syntax of assignment statements and class and method declarations are introduced in the first three assignments. Pointers are introduced in the next two assignments and are then heavily used in the remaining assignments. These remaining assignments cover topics such as arrays, inheritance, basic template usage, inheritance, operator overloading, copy constructors, parameter passing, file I/O, copy constructors, and debugging of pointer problems using GDB.

- **Standard Template Library**: The CS II course at Lafayette College covers the design and implementation of the Java Collections Framework using the textbook by Collins [5]. The Standard Template Library [18] is the C++ equivalent of the Java Collections Framework. This material is covered with three small assignments, similar to the ones on C++. The goal in these assignments is to highlight the similarities and differences between the various container classes. A standard assignment is to write a program that uses containers to store data but at the same time disallow the use of the more common container classes, e.g., list, vector and deque. In the course of these assignments, the students learn how to use templated classes but they do not learn how to write them.

- **use of tools for testing and maintaining software**: Students are given a few small assignments to help them understand how to use these tools. These assignments require the student to (1) write a makefile for a set of files that make up a program, (2) add, modify and delete files in a CVS repository, and (3) find the location and cause of run-time errors in a program. These tools are also used in the development and maintenance of the group project (see next item)

- **implementation of a medium sized group software project**: Students are assigned a semester long group project. Each group is composed of four to six students. The project is designed to teach the students about the various phases and necessary skills to successfully design and implement a medium sized software project.

- **event driven programming**: The group project requires that the students design and implement a graphical user interface. In the process, they also learn about event driven programming. The user interfaces are implemented using an object oriented interface to the GIMP Toolkit (GTK+)[10], called gtkmm. The interfaces are designed using a layout tool called glade. The advantage of using glade is that it can be configured to generate either C or C++ code. The students learn to use glade and gtkmm through reading the tutorials and online documentation including the API.

- **use-case object oriented design methodology**: The course uses the use-case driven design methodology developed by Jacobson, Booch and Rumbaugh [15] described in a text by Jacobson [14]. The description of the concept is augmented with examples from [21].

- **software testing of units and module**: Students learn how to test software by generating test plans from the requirements. Each student generates a test plan for the modules and classes that she implements and the group is responsible for generating a test plan for the overall project. Part of the grade for the project is based on the completeness of the test plan in addition to how well the project passes the tests.

In the past year, we have introduced the use of unit testing [2] as an integral part of the software development process. All programming assignments are required to be unit tested and students include the unit tests as part of their submissions. All lab assignments are now completed by pairs of students following the pair programming paradigm of extreme programming[1].

Much of the programming aspects and use of tools are covered by the small assignments given to the students. There are two assignments handed out per week for the first eleven weeks of the course covering C++ and the tools. This helps the students to quickly learn C++ and the use of the relevant tools quickly and enables them to work on the implementation of the project in the second half of the semester.
The assignments are completed by teams of two students working together.

Conceptual material is covered in the lectures and is augmented with readings from the textbook and supplemental material. The supplemental material includes books and papers by Norman [19], Brooks [4], McConnell [17], Dijkstra [6], Parnas [20], Beck [2, 1] and Boehm [3].

4. THE GROUP PROJECT

The group project is the focus of this course. The students acquire intangible skills through working on the project, e.g., interpersonal and time management skills. In addition, the project links together the many disparate pieces that they learn in the first half of the course, e.g., STL, pointers and software tools. Without it, the students are unable to see the connection between the material and software development. The project itself is one semester long, i.e., fourteen weeks and is broken down into multiple parts with submission dates throughout the semester as follows:

• requirements design (week 4): Each group submits a document specifying the use-cases that apply to their specific project. The project requirements are a set of general guidelines enabling each group to be creative about determining a unique set of requirements. The user interface is designed following the guidelines from Norman [19].

• architecture design (week 6): Each group submits a design for the system showing the general modules and the data flow paths between them. In addition, the design document also specifies the control flow between the modules when implementing each use case from the requirements document.

• detailed architecture (week 8): The design consists of the interfaces for all classes that will be in the system. After this has been submitted, each group can proceed to implement the system.

• test plan (week 10): Each group submits a plan showing how they plan to test their system and ensure that it satisfies the requirements and also does not have any major bugs. In addition, each member of the group submits a plan showing how they plan to test their units and modules. The focus on this part is on the data that will be used to test the modules, e.g. general cases and boundary conditions.

• final demo and presentation (week 14): The students are taught how to plan and script a demo as well as how to organize a presentation.

Examples of past projects include airline reservation systems, online course registration and most recently, a variation of the SIMS simulation game. The general requirements for each project are left deliberately vague so that the students learn how to refine and define requirements. For example, the requirement for the SIMS simulation is that each group develops a simulation game in the spirit of the SIMS. The game is required to have a GUI front end. Each team then has to decide (a) what type of game they wanted to implement, (b) what features would be supported. Part of the process including ordering the features so as to always have a coherent feature set even if all the features are not implemented due to time constraints. The students use an incremental development process and add new features as earlier ones are developed and successful tested. Rapid prototyping is fostered through an aggressive schedule of weekly demonstrations starting at the end of the ninth week of the course. At these demonstrations, each groups shows the current progress of their system to the other members of the class. At each demonstration, the groups also submit their current unit tests. This again emphasizes the integration of testing into the development cycle instead of leaving testing to the end as the students tend to do.

5. LESSONS LEARNED - EVALUATION OF THE COURSE

Software engineering has been taught in this format at Lafayette college for the past five years. Previously, it had been taught by a number of adjuncts and visiting faculty with a wide variation in coverage and expectations of the students. The dissatisfaction of both the students and the faculty led to the redesign of the course.

Approximately thirty to forty students take the course every year. The students organize into self-selected groups of between four and six students. The students mostly sophomore computer science and junior electrical engineering students.

In the first three years, we used the curses (actually ncurses) library [22]. The students were unhappy with being restricted to a character mapped screen interface. This past year, we introduced graphical user interfaces with the use of GTK+ and gtkmm. The students were much happier with the project even though more work was required in terms of learning how to write event driven programs using callbacks. This portion of the course and project turns out to be the most challenging part of the course. The software is still undergoing substantial development and change and the documentation is not easy to read for students who have only a year’s programming experience. We have had to add additional lectures to cover the particular implementations (which still changes substantially on a yearly basis) and provide example programs for the students’ use.

From a curriculum standpoint, the students emerge from the software engineering course much better prepared for the upper level courses in the department. Group projects requiring the use of C++, makefiles and CVS have been successfully completed without any additional instruction in operating systems, networks, artificial intelligence, databases and in the capstone course. In addition, the students have successfully designed and implemented GUIs in many of these courses. These upper level computer science courses have also been redesigned to eliminate the two weeks that used to be spent in covering the various tools. That time is now spent in covering additional material. The capstone course in computer engineering has also been extensively modified to include significantly more complex projects involving extensive use of software development tools and software engineering methodologies. This past year, the senior computer engineering class designed and built (from kits) two robots that navigated using optical sensors and communicated with each other through a wireless network and were programmed to work cooperatively to accomplish rudimentary tasks, e.g., delivering a piece of paper mail to a
specified location. The students developed requirements, a development schedule and test plans for the project. The instructors in that course could not have given a project that required extensive software development if the students had not learned the skills in the software engineering course.

One of the difficulties of using group projects is to ensure that every team member contributes to the project. One measure is to determine whether the student has learned sufficient software development skills to participate effectively in the implementation. At the mid-point of the course, the students are required to design, develop and implement a small individual project. The requirements of this program include (1) reading and writing data from files, (2) use of an STL container class to manage and store the data, (3) development of the appropriate unit tests and (4) a makefile to build the program. Examples of these individual projects include a small calendar program and filtering and processing data from a large data set. Typically, two or three students will not succeed at this point and they have to withdraw from the course. The skills that are tested and evaluated in this project are the ones that they will use in many of the upper level courses.

The students have increased confidence in their ability to tackle and complete large projects as well as learning new languages. Even though the workload is high when compared with other courses in the department as well as the rest of the college, the students feel that they have learned a lot and that it was worth the time that they spent on it. They also believe that their new skills help them when looking for internships. The extent to which the students enjoy the course depends very much on the “fun” quotient of the project. The online course registration system was regarded as more enjoyable than the airline reservation system and the Sim games had a much higher fun factor than any prior project. We intend to explore the games genre for future projects. Exit surveys of graduating seniors frequently reveal that they learned the most from the software engineering course.

We are a small department with a small number of students each year so there is only one section of the course that is taught every year. At this point, it is not possible for us to obtain a meaningful quantitative assessment as to the effectiveness of teaching these software development skills early in the computer science curriculum. The small individual project that is assigned at the mid-point of the course allows us to assess whether a particular student has mastered the software development skills up to that point. The student surveys provide unreliable assessment as to the utility of the course. A more reliable but still qualitative assessment was obtained from the instructors of the upper level courses both in computer science and computer engineering. In both departments, they report that the students have successfully used their software development skills to design and implement various projects. The computer science instructors have eliminated the portion of the upper level courses used to teach the requisite software engineering skills while the computer engineering instructors have introduced projects that require a greater degree of software development. Both departments agree that their respective curricula have benefited from teaching software engineering skills early in the curriculum.

From the instructor’s standpoint, the course can be time consuming. Grading the numerous small assignments is particularly onerous. In addition, the instructor has to meet with each group at two key points in the design process (requirements and architecture) to ensure that the student designs are viable. Overall, the course was very satisfying to teach because of the overwhelmingly positive response from the students.

6. CONCLUSION

This paper has described the software engineering course that is taught at Lafayette college to computer science majors in their second year and electrical and computer engineering majors in their third year. We have made the course one of the foundations of the curriculum and it has been successful in preparing the students for the upper level courses. The skills that the students acquire in software engineering have been successfully used in projects in those courses. The students overall feedback has been overwhelmingly positive. Our experience is that both the students and the computer science program have benefited from having software engineering taught early in the curriculum.

7. ACKNOWLEDGMENTS

This paper could not have been written without the encouragement and help of Bill Collins. Sesh Venugopal provided useful and timely suggestions that have improved the quality of this paper. In addition, the students who have taken the course over the past five years have made it a very rewarding experience.

8. REFERENCES


