Bridging Classroom Heterogeneity: A Software Engineering Course and Projects

Ani Nahapetian

In this paper, the process of bridging classroom heterogeneity in a Software Engineering course is discussed, using a real course as a framework for analysis. Specifically, this paper addresses issues when disparities exist in the same classroom, 1) between graduate and undergraduate students, 2) among students with a variety of programming skills and programming language familiarities, and 3) among student experience levels in software development. Additionally, the struggles of aiming for substance, while dealing with the perils of group work, are addressed.

This paper presents real and practical solutions to these challenges, including addressing issues with course content presentation, textbook selection, course projects, and graduate research opportunities.

Keywords
Software Engineering, Computer Science Pedagogy.

1. Introduction

Software Engineering courses exhibit a large degree of heterogeneity in terms of student abilities, skills, and motivation. This is especially true in the following situations:

1) the undergraduate course is co-located with a graduate version of the course;

2) some students have years of industry experience while others have learned to program only recently and have no programming experience outside of the classroom;

3) students approach the course with different expectations about what they will learn and the work they will do for the class;

4) students have different familiarities with programming languages and programming environments.

In this paper, I discuss the various approaches used to develop a Software Engineering course co-located with a graduate level advanced Software Engineering course.

Various conflicting interests presented themselves in developing the curriculum, and specifically the projects for these two courses. First, of course there was an issue with having a graduate level course and an undergraduate level course co-located. Secondly, I felt that a group project was a critical component to the course, but I wanted to avoid the hurdles faced when students worked in dysfunctional teams. The “I did all the work” syndrome could have a

Proceedings of the ACM-IFIP IEEIII 2008
Informatics Education Europe III Conference
Venice, Italy, December 4-5, 2008

significant effect on students' impressions of the course material. Thirdly, the students needed to be exposed to the basic concepts of Software Engineering, so they would appreciate that software development is much more than just writing code in a programming language.

In the rest of this paper, I present the challenges faced when preparing the course curriculum and syllabus for a diverse group of Software Engineering students in a single classroom. I present a survey of Software Engineering course syllabi and their use of textbooks and other reading material. I then present the various new and interesting approaches I employed in my classroom to address these issues.

2. Inherent Hurdles

2.1 Co-location of Graduate and Undergraduate Classes

For the first time in my department, a co-located Graduate Advanced Software Engineering course was offered at the same time and location as the undergraduate course. Though the undergraduate course had been offered previously the graduate course was entirely new, along with the brand new Masters degree program.

As would be expected this arrangement presented several hurdles. The first of which was the fact that the graduate students were expected to have previously taken an undergraduate Software Engineering course.

2.2 Variety of Programming Skills

There were a variety of programming skills in the classroom. My university carries out its introductory programming course mostly in Java. Though, one elective course on Programming in C in the UNIX environment is also offered. Additionally, there is a course that acts as a preparatory course to the undergraduate curriculum. This course has been taught in several languages including Visual Basic, Matlab, and Alice.

Despite this emphasis on Java, there are large numbers of transfer students who have studied C++ at their original college or university. Additionally, there are students who have a great deal of industry experience, who are most familiar with the programming skills acquired at their workplace. For example, in a classroom it is possible to find students well versed in SQL, but weary of their background in Java.

This presented several distinct options for the projects:

1) allow students to work in any platform they are comfortable, and then deal with compliance issues within groups as they arise;

2) impose one programming language on all the students;

3) offer a small set of popular programming languages, and compose the groups accordingly;

4) use a less known or proprietary programming language to level the playing field.

2.3 Dealing with Not Having Been in the Trenches

Software Engineering unlike other Computer Science fields is based on a large volume of empirical knowledge. Also to those with experience, critical issues and their solutions sometimes seem obvious. Students generally enroll in the course, once they complete their introductory programming series, and so they may take the course as early as their sophomore year.

In my Software Engineering course, there were students who had been working in industry for many years and were able to gleam quite a bit of benefit from discussions regarding practical aspects of Software Engineering, as compared with the students who had followed a direct path from high school to this course offering.

This presented a challenging dichotomy in the class. Some students, especially those with years of government industry experience, were very well versed in the techniques presented in the class, while others, including some of the graduate students, had no experience outside of the classroom and hence were learning the material from scratch.

Dealing with this issue was the most challenging hurdle faced in the course, as it spanned all the other disparities, including class rank and programming skills.

2.4 Avoiding Perils of Group Work while Learning to Work in a Team

There is research regarding the perils of group work [Waite04]. Invariably, when a large disparity between programming ability within a group exists, less experienced students defer an uneven amount of work to their more advanced programming classmates. Though opportunities to learn from each other are presented in these situations, students are not always the most adept at apportioning the work and aiding each other. As a result, learning opportunities are unevenly divided among the students, which can also lead to resentment.

On the other hand, working in a team is critical for a Software Engineering course. Industry demands teamwork and Software Engineering curriculum specifically addresses issues related to group software development. Students, during the job hiring process, are evaluated on their ability to work in or lead large and small teams. This is especially highlighted by our university's close proximity to large aerospace corporations. Note that software products in the aerospace industry are highly regulated and as a result use Software Engineering practices to a greater extent.

The key challenge here becomes creating opportunities for group work, while still overcoming the programming experience disparities among students.

2.5 Aiming for Substance

Software Engineering is often condescendingly considered a “pseudoscience,” and Software Engineering courses have a reputation for being the least technical of the Computer Science department’s offerings. E. W. Dijkstra is famous for saying “The required techniques of effective reasoning are pretty formal, but as long as programming is done by people that don't master them, the software crisis will remain with us and will be considered an incurable disease. And you know what incurable diseases do: they invite the quacks and charlatans in, who in this case take the form of Software Engineering gurus.” [Dijkstra]

The field of Software Engineering has advanced significantly since the time when this statement was made. Corporate hiring managers and university industry advisors strongly encourage and even demand that students complete Software Engineering courses at the undergraduate level.

Thus a key challenge for the course is to make clear to students that the seemingly basic topics that are the basis of a solid Software Engineering practice are truly valuable and have been developed at a great cost. Student resistance to such basic concepts as documentation and configuration management is prevalent and considerable. A key challenge is presenting the material so that students do not view them as unnecessary and time-consuming prescriptions.

2.6 Learning to Present Technical Material

Both industry and academia demand that students have strong presentation skills, including clarity, ease, and effectiveness. Software engineers are expected to engage in public speaking to handle code reviews, design reviews, requirement reviews, demonstrations, and more. Thus incorporating presentation opportunities became an important consideration in my Software Engineering course development.

3. Textbooks

3.1 Not for Inexperienced Software Developers

Software Engineering literature is often written for engineers with industry and large scale software development experience. For a large majority of the undergraduate and even graduate students in my class, this is not a correct assumption. Students often take Software Engineering right along with their Computer Architecture and Computer Networking courses. There is no large time gap between the students’ initial introduction to programming and their discussion of Software Engineering. Additionally, Software Engineering curriculum has a focus on managerial duties that may be new to even experienced programmers.

The search for a textbook or other reference material is a challenge that needs to consider student experience levels, while still aiming for the best and the latest empirical approaches that Software Engineering has to offer.

3.2 Survey of Online Courses

Software Engineering courses are more likely to not follow a single text, than any other core undergraduate Computer Science course. Table 1 summarizes the textbook information found in six syllabi found on the web after some reasonable amount of web searching. 2 out of 6 online course syllabi did not have a course textbook. None of the online course syllabi used a single textbook. Additionally, 3 out of 6 used published and web articles as part of their curriculum.

Table 1. Software Engineering Textbooks for Various Universities

Reference	University	Offering	Text Type
Text(s) Used			
[CSUN]	CSUN	Fall 2007	Multiple texts
1. Ian Sommerville, *Software Engineering, 8th Edition*, Addison-Wesley Longman Publishing Co., Inc (2007). 2. Dan Pilone and Neil Pitman. *UML 2.0 in a Nutshell, 2nd Edition*, O'Reilly Media, Inc., (2005).			
[CalPoly]	Caly Poly San Luis Obispo	Spring 2007	Articles
1. Supplemental reading			
[Pomona]	Pomona College	Fall 2007	Single text, supplemented with articles
1. Steve McConnell, *Code Complete, Second Edition*, Microsoft Press (2004). 2. Supplemental reading			
[UCLA]	UCLA	Winter 2008	Multiple texts
1. Steve McConnell, *Code Complete, Second Edition*, Microsoft Press (2004). 2. Roger S. Pressman, *Software Engineering: A Practitioner's Approach, 6th edition*, McGraw Hill (2005).			
[Washington]	University of Washington	Spring 2007	Multiple texts
1. Steve McConnell, Software Project Survival Guide. Microsoft Press (1997). 2. Andrew Hunt and David Thomas, *The Pragmatic Programmer: From Journeyman to Master*, Addison-Wesley Professional (1999).			
[Cornell]	Cornell	Spring 2008	None
1. None, only recommended texts			
[Berkeley]	UC Berkeley	Spring 2008	Articles
1. None, only recommended texts			

4. Project Framework

After considering the course challenges presented in the earlier sections, I chose to address them with the following approaches.

4.1 Course Content Presentation

In an effort to bridge the disparities that existed among the students, I took several different approaches to course content presentation. The various forms of imparting information not only reinforced concepts, but also enlarged the group that was able to follow the material.

Specifically, I employed a comprehensive Software Engineering text [Sommerville07] and also incorporated articles, some from a classic Software Engineering book [Brooks95]. Additionally, I used palatable, but also very powerful, articles on various important coding topics. I emphasized a project development atmosphere in the course, so students would have the opportunities to flex their software development muscles. Finally, I incorporated student presentations into the mix, so that students would be able learn from each others experiences and activities, as well as from the text and me.

Another essential component became classroom discussion. I found that students, even despite their inexperience, still had some basis on which to make valuable comments and well posed arguments. Additionally, the more experience students were able to provide an invaluable perspective about the field and the practice of Software Engineering, that all the students found beneficial. For example, the class was able to hear about how several different local companies' approach the code review process, with experienced students taking on a role similar to that of a guest industry speaker.

4.2 Split Project Nature

I divided the course project into four segments: requirements, design, code, and testing. The first two segments, requirements and design, were carried out as a team of three or four students. The students developed a software project idea and prepared a requirements document for their project.

The next segment of the project was to develop a design for the project. As a group, a high level component based design was developed, where each team member eventually worked to develop a component of the design.

The novelty of my course syllabus lies in the splitting the team for the code development segment. To address the heterogeneity challenges presented in the earlier sections, students wrote their own code without being able to rely on their partners for help. Individual code development with the aim of combining the code components into a larger project is common in industry. It also clarifies the need for practices such as documentation, proper interfacing, component testing, integration, and more. It also addresses the inherent differences between student programming skills which lead to the perils of group work discussed earlier.

In an effort to give students a technical software development project which they would work on individually, I considered the following options before settling on the split project idea. I considered having students 1) edit legacy code, 2) develop a small pieces of open source software, and 3) complete each others code.

4.3 Requirements, Design, and Code Reviews

In an effort to provide technical presentation practice, the first three segments of the project culminated with presentations, namely requirements review, design review, and code review. Students presented as a group, or individually in the case of the code review, according to the format of industry review meetings. All members of the class were expected to comment during the review, and students had to implement the instructor's comments in the next phase of the project.

The reviews provide a forum for students to learn technical presentation skills, to engage in the common practice of software reviews, and to learn to handle questions and comments on their work. Additionally, it is an opportunity for students to learn their role as a member of a review committee.

4.4 Reading Research Material

To encourage students to approach Software Engineering not only as a topic to learn, but also as an accessible field about which to do research, I decided to expose graduate and undergraduate students to Software Engineering research literature. The topics in Software Engineering literature tend to be more palatable to wider audiences than other Computer Science fields. Additionally, contributions from software engineers in the field are both practical and informative.

Students read and analyzed several articles throughout the semester. The first article was used as part of a take home midterm, to ensure that students read the article fully and were given an opportunity to express their ideas on the material. The article was a publication on code reviews, thoughtfully written by an industry software developer in very plain language [Wiegers98]. It was the first Computer Science research article that many students had ever read, and since the material presented in the article was relevant to the students' lives and also easy to understand, I believe that it helped form a positive impression of Computer Science research and became a feasible entry point into the world of research. This exercise was the best received assignment during the semester.

Throughout the course, students were asked to read selected Software Engineering excerpts from F.P. Brooks' famous text The Mythical Man Month [Brooks02]. Though the text is a classic and an essential part of Software Engineering, it did not give the students a sense of the current interests in the field, the way the research articles did.

4.5 Graduate Research Project

The standard approach for handling graduate course work, in a largely undergraduate class is to give extra work to the graduate students. However, I had reservations about that approach, since I felt that the graduate students might get bored, that it was the not be the best use of their time, and that in turn might detract from the undergraduate learning experience.

In the end, I gave the graduate students a semester long research project assignment in place of the final exam. The graduate students participated in all other parts of the course, including the projects and the midterm. Outside of the classroom on an individual basis, the graduate students and I discussed the research project, and the work was presented to the

entire class at the end of the semester. The work carried out for the research project involved choosing a research idea in Software Engineering and working towards and preparing a conference level research article on the idea.

4.6 Scheduling Graduate Work

I grappled with the idea of ending the undergraduate class session early to introduce and engage in discussions about graduate concepts. Though, this presents an opportunity to present graduate level work, it does cut the undergraduate lecture time short. Having a small number of graduate students in the course permitted some of the graduate needs to be addressed outside the classroom. In the end, I decided to hold combined undergraduate and graduate lecture sessions, but to have additional weekly meetings with the graduate students.

Through the process of preparing a graduate research paper, a great deal of interaction occurred between the graduate students and me. I believe this greatly enhanced their course experience. As the graduate program becomes more popular, this may become more difficult to do.

I had one request from a graduate student to do the course software development project individually. I did not agree to this, as the graduate students need opportunities to be involved in a software development group, both for their benefit and that of the undergraduates.

4.7 Final Opt-Out

Due to the split project nature of the course, students do not necessarily need to consolidate their components into the final software project as they envisioned. However, since they should each have a working component of the system, the option to integrate the components is presented to the students. As a reward, the students are allowed to opt out of the final, upon successful completion of their original project proposal.

Some students were highly motivated to take this option, as it provides the satisfaction of completing a large software engineering project. Additionally, it allows the motivated groups to obtain a deeper perspective about the software development process.

Though making the integration part of the project mandatory is enticing, it presents a problem when some group members do not fully accomplish their component requirements. This puts the other group member in a difficult situation. I find that the optional, final opt-out approach is able to handle this situation more fairly, by giving dysfunctional teams a break by taking the final exam instead.

5. Conclusion

In this paper, the process of bridging classroom heterogeneity in a Software Engineering course is discussed. A real course is used as the framework for addressing classroom disparities, in terms of class ranks, programming skills, and industry experience levels. Additionally, approaches to avoid the perils of group work are presented, given this level of classroom heterogeneity.

6. Acknowledgements

I would like to thank Professor Miodrag Potkonjak of UCLA for an interesting discussion regarding Software Engineering textbooks, which greatly contributed to development of my ideas on the subject. Also, I would like to thank Theodor Soneriu for collecting some of the data used in Table 1.

References

[Berkeley] Berkeley Software Engineering Course Website. http://inst.eecs.berkeley.edu/~cs169/sp08/doku.php?id=info

[Brooks95] Brooks, F. P. The Mythical Man-Month (Anniversary Ed.). Addison-Wesley Longman Publishing Co., Inc (1995).

[CalPoly] Cal Poly Software Engineering Course Website. http://www.csc.calpoly.edu/~djanzen/courses/307S07/

[CSUDH] California State University, Dominguez Hills Software Engineering Course Website. http://www.csc.csudh.edu/ani/courses/2007Fall/csc481-581/csc481-581.html

[CSUN] CSUN Software Engineering Course Website. http://www.csun.edu/~twang/380/

[Cornell] Cornel Software Engineering Course Website. http://www.cs.cornell.edu/courses/cs501/2008sp/

[Dijkstra] EWD 1305: Answers to questions from students of Software Engineering. http://www.cs.utexas.edu/users/EWD/transcriptions/EWD13xx/EWD1305.html.

[Pomona] Pomona College Software Engineering Course Website. http://www.cs.pomona.edu/classes/cs121/

[Sommerville07] Sommerville, I.. Software Engineering, 8th Edition, Addison-Wesley Longman Publishing Co., Inc (2007).

[UCLA] UCLA Software Engineering Course Website. http://www.cs.ucla.edu/classes/winter06/cs130/syllabus.html

[Washington] University of Washington Software Engineering Course Website. http://www.cs.washington.edu/education/courses/403/07sp/syllabus403.html

[Waite04] Waite, W. M., Jackson, M. H., Diwan, A., and Leonardi, P. M. Student culture vs group work in computer science. SIGCSE Bull. 36, 1 (Mar. 2004), 12-16.

[Wiegers98] Wiegers, K. 1998. The seven deadly sins of software reviews. Softw. Dev. 6, 3 (Mar. 1998), 44-47.