An Introductory software engineering course that facilitates active learning

Stephanie Ludi
Swaminathan Natarajan
Thomas Reichlmayr

Follow this and additional works at: http://scholarworks.rit.edu/article

Recommended Citation
Ludi, Stephanie; Natarajan, Swaminathan; and Reichlmayr, Thomas, "An Introductory software engineering course that facilitates active learning" (2005). ACM, Accessed from http://scholarworks.rit.edu/article/1203

This Article is brought to you for free and open access by RIT Scholar Works. It has been accepted for inclusion in Articles by an authorized administrator of RIT Scholar Works. For more information, please contact ritscholarworks@rit.edu.
An Introductory Software Engineering Course that Facilitates Active Learning

Stephanie Ludi
Rochester Institute of Technology
134 Lomb Memorial Drive
Rochester, NY 14613
(585) 475-7407
salvse@rit.edu

Swaminathan Natarajan
Rochester Institute of Technology
134 Lomb Memorial Drive
Rochester, NY 14613
(585) 475-4663
sxnvcs@rit.edu

Thomas Reichlmayr
Rochester Institute of Technology
134 Lomb Memorial Drive
Rochester, NY 14613
(585) 475-2852
tjrese@rit.edu

ABSTRACT
At the Rochester Institute of Technology, the undergraduate introductory software engineering course has been redesigned from a lecture-lab format to a project-centric studio format. The new format blends the lecture material with the project work. As a result, students drive their own learning experience based on scaffolding created by the course design. The challenges faced and the techniques and strategies utilized in the planning and delivery of the course will be discussed, including the utilization of online learning support infrastructure. This paper presents instructor experiences, analysis of student feedback, lessons learned and recommendations for other educators considering an active learning approach for their courses.

Categories and Subject Descriptors
D.2.9 [Software Engineering]: Management-programming teams.
K.3.2 [Computers and Education]: Computer and Information Science Education: curriculum.

General Terms
Management

Keywords
Software Engineering, Active Learning, Cooperative Learning, Student Teams, Student Centered Instruction

INTRODUCTION
The learning paradigm for computing courses has remained almost intact for the last 25 plus years. Under this paradigm, students in a course meet weekly for lecture and then for a lab, but experience has shown that students often have trouble “connecting” what is taught in class with what is required from them in lab. While research eloquently shows the many advantages of the studio format, implementing an active, studio-centric learning environment presents many challenges. A balanced collaboration between students and instructor is required to achieve the full potential of this approach, with the instructor’s role shifting from knowledge disseminator to learning facilitator. Students must accept more responsibility for their own learning and adapt to a classroom style that is different from that experienced in previous courses.

The introductory software engineering (SE) course at RIT (SE361) is a sophomore-level required course for software engineering, computer science and computer engineering majors. RIT operates on a quarter system, consisting of 10 weeks of classes followed by an “exam week” at the end of each quarter. Fifteen to twenty sections of the course with up to 20 students each are taught during each academic year, by 8-10 different instructors. Because of this, the structure and delivery of this course are heavily streamlined. Until the Fall of 2003, this course had been delivered in the classic lecture-lab format with three hours of lecture per week and a two hour lab session.

The primary objectives of our introductory SE course are twofold: First, to give students the opportunity to experience teamwork while working on a term-long project that spans a full development lifecycle, applying effective SE practices, and following a process discipline. And second, to expose students to fundamental concepts of software design and process such as lifecycle models, design patterns, testing and estimation.

Among the problems we experienced with the classic format was that students perceived lecture and lab as two disparate experiences. While the lab allowed them to participate in software engineering activities, the lecture “only taught them a few basic SE principles”. Although lectures took significant time per week, students did not see them as adding value to their projects. Lectures became monologues where instructors spoke about subjects the students perceived as tangential to their lab project. Students tended to learn the material for the sake of exams, rather than relating it to their project work and as being fundamental to understanding software engineering.

The even more serious impact of the perceived disconnect between lecture and lab was its influence on perhaps the most important
desired course outcome: understanding the need for process discipline and sound SE practices. It was the lecture that discussed the underlying rationale, but this was viewed as “theory”. The labs were graded on a defined set of deliverables, which were viewed as outputs to be produced, without much thinking as to why they were needed.

Over the past two years, we have focused on strategies for the improved delivery of this important foundational course in our curriculum. Our first step was to move the student team project component from a heavy-weight, waterfall development model to an iterative one using a lighter-weight, more agile approach to process [1]. This model allowed students to develop their projects in increments and provided the opportunity to experience a more realistic environment where changes are introduced during the course of a product’s development life-cycle. It also provided the instructor with more opportunities for feedback that could be incorporated into future development iterations.

Although the move to an agile project component helped generate enthusiasm among the students, a disconnect between lecture and lab still lingered. To address this issue, we designed our course in a manner that would move students from passive lecture spectators to engaged, active learners, and more effectively integrate the course’s project component. The redesigned course structure and its implementation results are the focus of the remainder of this report.

1. Towards a More Active SE361

The Software Engineering Department at RIT has committed itself to engaging students and creating an active classroom environment through systemic change in the courses that make up its curriculum. During the summers of 2002 and 2003 department faculty attended Project Catalyst workshops at Bucknell University [2,3] designed to introduce active learning techniques and aid in course re-design that adopts student centered instruction. Student-centered instruction shifts the responsibility of learning from one that is entirely the instructor’s to a shared responsibility between student and instructor [4]. Student-centered instruction requires students not only to passively listen and take notes, but to engage in such higher-order thinking tasks such as analysis, synthesis, and evaluation [7]. This approach transfers the responsibility to students. Students bear the responsibility for constructing their own knowledge and participating in activities that require social interaction between other students and the instructor.

Our goal in redesigning the course was to focus on shifting the traditional lecture portion of the course more towards student-centered interaction as shown in the active learning spectrum in Figure 1. The solid arrows represent the previous SE361 lecture and lab, and the dashed arrows represent goals for the re-designed course.

The active learning spectrum is defined by the following four phases [2]:

- **Active Learning** – students actively participate in learning using individual activities with the opportunity to interact with other students.
- **Collaborative Learning** – more interaction with other students, informal group activities.
- **Cooperative Learning** – formal, structured team activities.
- **Problem Based Learning** – problem situations drive learning activities on a need to know basis.

**Figure 1. Active Learning Spectrum**

Having just gone through a major re-design of the course’s project component in the previous year, we decided to leave the format of the project intact. Minor tweaks were made to make the project’s problem statement more open ended and provide for creative approaches by the student teams [6].

In addressing the lecture component of the new course we had the benefit of tapping into the experiences of another course in our program that had recently undergone a similar redesign [5]. While that course transformation traversed the active learning spectrum all the way to problem based learning, the experiences gained making lectures more student centered at the active and cooperative learning levels were very helpful here.

2.1 The Redesigned Course

The first step in our redesign was to align the lecture material with the project’s flow to achieve their integration. Because so much of the lecture material is needed so early in the course, this proved to be a significant challenge. As mentioned earlier, the project had recently been modified to use an agile, iterative development approach.

The first project iteration, R1, was due at the end of the 5th week, and required basic implementations of the major functionality. The second iteration, R2, due at the end of week 8, added a graphical user interface and included nearly all of the product features. The final iteration, R3, was basically a perfecting release that included thorough testing and bug-fixing, and cosmetic improvements such as help facilities.

This schedule meant that R1 would itself involve a complete software engineering lifecycle, including requirements definition, planning and process choices, architecture and design, integration and testing, and proper configuration management. Therefore we had to provide the student teams the theoretical foundations in all of these areas within the first five weeks of lecture. In addition, students were also being exposed to a new development environment and configuration management tool [8]. Achieving this fast-paced approach to learning while moving away from instructor-centric lectures was particularly challenging.
To accomplish this, we adopted a three-pronged strategy. Firstly, a two-pass strategy was adopted in the coverage of theory material. On the major topics such as design, testing and processes, the first pass focused on the basic concepts and understanding needed for the successful execution of project activities. The second pass covered more advanced topics such as design patterns, dynamic modeling, interface design and evaluation, alternative integration strategies etc., which would be needed for the second and third parts of the project.

The second part of the strategy was to design in-class activities on each topic that would enable students to apply the concepts and familiarize themselves with the practices during class. Some of these activities were related to the project, while others were standalone activities. These activities served as a bridge between theory and practice, and ensured that they gained the competences needed for the project.

The third part of the strategy was to change the role of lectures from covering material to helping students integrate the material and provide perspective. The students were expected to prepare for class by reading relevant sections from the textbook and other sources, and the lectures and in-class activities completed and consolidated this learning process, rather than relying on lectures as the primary source of knowledge.

### 2.2 Course Structure

The resulting course structure consisted of three components: lectures, in-class activities and project work. We defined weekly milestones for the project, and matched weekly lecture topics to the milestones. The course was scheduled so that there were two two-hour sessions each week. We conceptually divided each session approximately into four half-hour blocks, and allocated these blocks as either lecture, activity or project blocks, with a goal of having roughly equal numbers of each of the three over the course. Most sessions had at least two types of blocks. The idea of this was to facilitate multiple modes of instruction and learning. The resulting course plan is shown in Table I.

![Course Plan Table](image)

The lecture component is relatively high during weeks 2-4 (typically two lecture blocks per session), and tapers off significantly towards the end of the course, with the focus shifting mostly to the project.

The project concept we have been using during the initial offering of the course is to create a game that helps first and second graders to learn math. The target audience is deliberately chosen to be different from the students, to encourage reflection on user characteristics. Each team can design and implement their own game, as long as it serves the customer’s need of helping children to learn math. Students manage all aspects of teamwork themselves, and cooperation between teams is encouraged, as also reusing software from outside sources (with acknowledgements).

### In-class activities

- Hands-on tutorials on tools usage, for Eclipse [8] (the development environment), configuration management, and JUnit [9] for unit testing. The JUnit tutorial also shows them how to use a test-driven development strategy if they so desire.
- In-class exercises on writing requirements statements, drawing state diagrams.
- For architecture and design patterns, the instructor and students work through a design example together as a class.
- Role-play in conducting an in-class inspection meeting, where students review sample code to find defects and improvement opportunities.
- Project-related team activities, including creating paper prototypes of interfaces, and later teams evaluating each other’s interface based on the prototypes.
- Class discussions on integration strategies that were actually used and those that should have been used in the team projects.
- Case studies of realistic software engineering situations as a course wrap-up. Students work on these first individually and then in teams, followed by an in-depth class discussion.
- Team reviews of the course material before final exams, where each team creates questions and answers on each of the topic areas covered.

### Course infrastructure

The third part of our strategy involves students being able to drive their own learning processes, rather than relying exclusively on lectures. To facilitate this, we create extensive online materials to help students structure their learning.
“Topic Outlines” were created for each area, that:

- Identify the learning objectives for the area.
- List the various subtopics to be covered, and provides sources for learning each of them. This includes the specific pages in the textbook to read, as well as website URLs that they can go to for additional information if desired.
- Provides a set of “Guiding Questions” that help them to evaluate their learning, and to prepare for exams.

The idea is for the Topic Outlines to serve as a complete self-study guide. For the project, we create a week-by-week list of deliverables, with detailed instructions for each deliverable and project activity. These instructions are written in such a way as to avoid being specific to the project content, so that they can be reused with different projects. Thus each project team knows in advance exactly what was expected of them throughout the project.

Predefined document templates are provided for most project deliverables. Each document is carefully engineered to minimize the work needed to complete it, while simultaneously ensuring that there is due attention paid to all the key content and process discipline aspects. This includes an activity tracker spreadsheet that contains the various project activities and deliverables, where they enter completion status, planned effort, task allocation and later actual effort information on a weekly basis. A “process grid” is created that provides a range of practices of different levels of formality for each process area, and teams can select the level of process they considered appropriate. The objective of this is to shift their mindset from “writing reports and documentation for the process they considered appropriate. The objective of this is to shift their mindset from “writing reports and documentation for the instructor” to “creating artifacts to track project progress and decisions”, where hopefully they will see more clearly the value that each document is adding to their work.

Detailed information. A “course map” diagram has been created to help them navigate the course structure, as shown in Figure 2.

The goal in the course materials and their structure is to move away from an instructor-centric concept where the students are constantly dependent on the instructor to lead their learning, tell them what to do and when to do it.

While redesigning the course was a challenge, our work was facilitated by our department’s superb facilities. Each class meets in a studio classroom that provides each student with a computer that the instructor can control from an instruction workstation. In addition, to enable student teams to conduct team activities, the departments operate 11 team “breakout rooms” each equipped with a computer hooked to a ceiling-mounted projector, Ethernet connections, and comfortable seating for six.

3. OUTCOMES

In order to assess the new course format, the students were surveyed at both the course midpoint and at the end of the quarter. The survey was administered online to students enrolled in the course during the Fall, Winter, and Spring quarters. The survey administration was piloted during the Fall quarter, while the Winter quarter offered the most representative sample of students from 11 sections of the course.

During Winter quarter, 126 students completed the survey. The survey asked the students to supply demographic information, but concentrated on gauging the effectiveness of the course structure, project, course elements and student preparation. The students also had the opportunity to comment on aspects of the course that they valued and aspects of the course that should be revised.

Since the structure of the course has been designed to rely less on lecture and more on the project, we were interested in gathering feedback as to the pace of the course and the proportion of lecture to project time. While the course is fast-paced, 69% of the students responded that the pace of the course was just right. By comparison 20% of students rated the pace as “Too fast” and 10% of students rated the pace as “Too slow.” During the course many students commented informally that the project was very important to them and that they wanted more course time devoted to the project. The statistics supported these comments. In terms of lecture, 47% of students were content with the amount of lecture in the course while at the same time 47% of students wanted less lecture. In terms of the effectiveness of lecture, 31.7% viewed lecture as being either Helpful or Extremely Helpful while 34.1% found lecture to be Moderately Helpful.

The students spoke strongly in terms of the extent of working on the project in class. While 28.6% of students felt that the amount of class time devoted to the project was adequate, a 65.1% of students wanted more class time spent on the project. The students responded to the technical and procedural aspects of the project. In terms of the overall project experience, 79.4% of students found the project itself to be either Helpful or Extremely Helpful. Regarding the deliverables that represented the application of the various Software Engineering concepts covered in the course, 77.8% of students found the various project deliverables to be either Helpful or Extremely Helpful. Such results reinforce the positive impact that the significant course-long project provides in terms of technical and Software Engineering interest.
In terms of the project, a sample of student comments includes:

“The best aspects of the course are being able to go through an entire project's development process, from requirements to final release. Taking a position in the team and being responsible for those roles throughout the entire quarter helps my understanding of the software development process. I learned a lot from this course.”

“I enjoyed working on the project most of all because it gave us a view of real world software development.”

“Project provides a way to get practical experience and feel like you are producing a real project rather than a useless piece of code for some trivial application.”

“The project was hard but it was a lot of fun!”

Since considerable effort was devoted to designing the course infrastructure, we wanted to assess its effectiveness. The results showed that much of the course infrastructure was not perceived as helpful to the students. Only 48.4% of students found the course topic objectives to be either Helpful or Extremely Helpful. Besides the objectives, each major course topic’s outline presented a structured view of topic concepts (including references to course readings). Only 50.8% of students felt that the topic outlines were either Helpful or Extremely Helpful. Only 27.8% of students viewed the required readings as either Helpful or Extremely Helpful. This result reflected the perception that students relied on slides for studying rather than reading the text either before class or before a quiz/exam. 62% of students stated that they were prepared for class (in terms of reading) either Rarely or Never. However when the preparation corresponded to a project deliverable, the 97.6% of students were prepared either Usually or Always. When the students see direct value for the preparation (when directly applicable to the project), then the level of preparation is significantly higher. The students want to see clear value in what they are asked to read and do.

The survey contained areas for open-ended comments about the course in any aspects that the student wished to comment. Some student comments regarding the merit of the course include:

“All the group presentations helped us get different perspectives on everything.”

“I liked the project and found it very helpful in helping me understand the class as a whole. I liked having set times when things were due so I could plan ahead and be ready for them.”

“I like the way this course is structured as compared to the previous CS (Computer Science) courses. Since my professional electives are going to be primarily SE instead of CE (Computer Engineering), this course provided me with a solid foundation in my opinion.”

Other comments offer constructive feedback in terms of where to focus efforts in further offerings:

‘Lecture didn't always help with project activities, more in class activities would be helpful.’

“Need more project time; possibly make the class worth more like 5 or 6 credits and add an hour or two to class time.”

The course instructors discussed the survey results, and the feedback was generally consistent with what was observed by instructors in the classroom and with student comments in class. Nearly all of the current course instructors had taught the course in previous renditions. The general perspective was that the new course design had fewer student complaints than previous course offerings. Fewer project failures were also observed.

4. CONCLUSION

The new approach of integrating the lecture and project work has been very successful in reducing the perceived disconnect between the theory and the practice. In-class activities are proving to be both popular and effective in consolidating learning. The attempt to move towards self-guided learning has been markedly less successful, and students are still relying almost entirely on the lecture to learn the concepts. Future efforts will work towards identifying the issues and strategies needed to more effectively address the reliance on lecture.

The new format has certainly succeeded in its goal of more active learning. Only about 30% of class time is spent in classic lecture mode. The strong linkages to the project have increased the effectiveness of the project as a learning vehicle, and made the course more clearly project-centric.

5. REFERENCES


