Paper ID #30235

A Course as Ecosystem: Melding Teaching, Research, and Practice

Dr. Edward F. Gehringer, North Carolina State University

Dr. Gehringer is an associate professor in the Departments of Computer Science, and Electrical & Computer Engineering. His research interests include computerized assessment systems, and the use of natural-language processing to improve the quality of reviewing. He teaches courses in the area of programming, computer architecture, object-oriented design, and ethics in computing.

A Course as Ecosystem: Melding Teaching, Research, and Practice

Abstract

We often compartmentalize our academic life into the areas of teaching, research, and practice. In fact, there are many synergies to be realized by treating a course as a complete ecosystem. This means enlisting students in the course to work on projects to improve the course, and projects to help the instructor in research. Managing these projects can even give instructors a taste of what it is like to manage projects in industry, giving them experience applying course concepts in the field. Projects within the course can lead to independent-study projects, or even theses. With a little bit of ingenuity, this strategy can be employed in courses from the introductory to the graduate level.

Keywords: software engineering, instructional technology, mentoring, independent study, teaching assistants

1. The concept of a course "ecosystem"

A conventional way to look at a course is as a piece of a curriculum. It has certain learning objectives, and if students have achieved them by the end of the academic term, then the course can be considered a success. However, there is another, more active, way to view a course: a course is an opportunity to direct students in their learning. This is a more powerful perspective, because it suggests that students learn by *doing*, not necessarily following the same path as other students. Also, by following different paths, students can make their own contribution to the instructor's "ecosystem" of teaching, research, and service.

Most departments will allow a newly hired faculty member to teach at least one course in their specialization. The students who take such a course will have acquired enough competence to write software components for a system, or can carry out analyses that are useful in research. When planning such a course, think about projects that can

- be used in your research (e.g., can help collect or analyze data, or serve as test beds for research ideas),
- provide the students with the background that they need in order to begin research with you (e.g., become familiar with your experiments, your data, or your analysis software),
- keep the students engaged with the material after the course ends (e.g., as peer tutors or project mentors),
- help improve the course (e.g., by creating new active-learning exercises over the material, or scoping out new technological developments that could be incorporated into the course), and
- keep you in contact with current industrial practice (e.g., by serving as a scrum master or training others in the practice).

This way of looking at a course reflects a subtle, but important, difference in devising course projects. The question is not, How can I specify projects that will familarize students with the course content? but rather, How can I design projects that will help students find their role in promoting their own learning and that of their classmates? This latter view gives the instructor a vested interest in the success of projects, because they benefit not only the students, but also the instructor's own teaching and research.

The idea of an "ecosystem" is not new; in fact, senior design is often taught that way: students work on some industrial projects and some faculty research projects, and the goal is to produce working prototypes that will build relationships with sponsors and strengthen the course. But it's not apparent at first glance that many other courses can be viewed in the same way, as self-nurturing ecosystems. This paper contains multiple examples of where ecosystems have been successfully constructed in advanced courses. The concept may also be usable in introductory courses, but admittedly there are hurdles, such as the amount of direction each student or team would require in order to produce useful artifacts.

The rest of the paper is divided into two parts. First, the author describes his own experience and recounts lessons learned. The second part explores how these techniques might be applied to courses in other areas of software engineering.

2. A Case Study: Object-Oriented Design and Development

Early in his career, the author developed a course in object-oriented programming and concepts of object orientation. The course in question is an advanced undergraduate and masters-level course. About twenty years ago, he realized that students in the course could develop web applications that were useful in managing the course. Early examples included software for maintaining a database of homework and exam questions contributed by other instructors, a "signup sheet" to allow students to choose projects over the web, and a peer-assessment system to allow students to review each other's work.

In the years after 2000, the open-source movement took off, and computer-science faculty started to incorporate open-source software (OSS) projects into their courses. Open source gave students a way to make contributions that would be used far beyond the course for which they were written. Often they were worth citing on a résumé. The author's course acquired more of an emphasis on development. Students, who had formerly been allowed to choose their own topics for a project, were now required to contribute to an open-source project, using the skills they were learning in class. To facilitate this, the author sought out OSS projects that were looking for student contributions. Sponsors such as JFreeChart, Sahana, and OpenMRS contributed specifications for student projects.

It was never possible to get outside sponsors for all of the projects needed for the class. By 2011, the class had grown to over 100 students, which meant that 30 to 35 distinct projects were needed for each assignment. An outside sponsor would usually provide only about three projects. So the slack had to be taken up by internal projects. Several years earlier, we had open-sourced our Expertiza peer-assessment system, and we were adding features rapidly. Our web-based system had been used at more than a dozen other institutions, so it did give students the experience of writing software for a real user base.

This development benefited students in several ways. First, assignments were “real world” in the sense that they had real users, but also because they were based on modifying and extending existing code. This is very different from starting a project from scratch. It’s necessary to read code that has been written by other students, which gives students an appreciation of the importance of coding standards and good commenting. There are plenty of opportunities for refactoring. Documentation is important, because that’s how future teams will come to understand the design. Sometimes students whose projects are “close to” being usable are given extra time and one or two extra points for fixing these deficiencies. This is a more authentic practice, and keeps students more accountable, but it wouldn’t happen in a regular class, where it would be extra work for students and instructor with no offsetting benefit.

Second, students’ contributions benefited the experience of other students, because the software enhancements could be used to manage student contributions in later semesters. A couple of examples will serve to illustrate. One course project was to implement a task list, which helped students keep track of upcoming submission and review deadlines. Another project created a system for bidding on projects. This was important because it eliminated the mad scramble for projects that began when they were announced. The first thirty or so students would reserve all the topics, and later entrants were placed on a waiting list, which would typically never clear. Students then had to seek out others who had been able to reserve topics, and join their team. Bidding assignment now coalesces students into teams based on their preferences, before assigning teams to topics, thus eliminating the rush.

Third, the fact that good course projects could be incorporated into the software application gave the instructor intrinsic motivation to review the projects carefully. If he did this well, good projects could be merged, and the management software would improve. If he was careless, the same projects would likely have to be repeated in a later semester. In addition, some of the projects were used to derive data for his students’ research. It was essential for them to be implemented correctly. Likely there would be no other way to motivate the instructor to exercise such meticulous oversight of projects that students do in a course.

Thus far, we have been describing projects done inside the course. In the early days, these projects added new features, but more recently, refactoring and testing projects have become more common. There are also projects of larger scope, done for independent-study courses or theses. They also benefit the course, because they improve the peer-assessment software used in the course. One independent-study project let students form teams by inviting others to join their team, and having the invitees then accept the invitation. Another project added a tabbed view for assignment creation. A third provided an anonymized view so that we can use live data in demos without revealing any student names. Four of these projects have led to conference papers co-authored by independent-study students [1–4] and five to workshop papers [5–9].

The first masters thesis added features to support peer-reviewing student contributions to a wiki textbook [10]. This involved sequencing review of chapters so that prerequisite chapters would be written and reviewed before chapters that depended on them. The first Ph.D. dissertation [11] involved the use of natural-language processing to evaluate features of reviews [12], e.g., their tone (positive or negative), whether they seemed to be making comments relevant to the work, or whether they were mostly just saying that the reviewed work was good or bad. The second Ph.D. [13] looked at aspects of peer assessment, such as improving rubrics [14], separating formative and summative rubrics [15], and using reviewer competence metrics as weights in calculating peer grades [16]. A third Ph.D. focused on the software-engineering aspects of the work [17], such as analyzing the design mistakes students made in their projects [18], measuring the impact of different strategies for test-driven development [19], and employing bots to give formative feedback to students on their code before they submit their project [20]. A current Ph.D. student is studying machine-learning approaches to measuring review quality [21]. It is important to note that none of these projects could have taken place without the use of the peer-assessment system in my object-oriented development course: we would not have had review data to work with, and we would not have had student projects to measure.

2.1 Managing independent-study projects related to the course

Independent-study projects have been essential to sustaining this ecosystem. Some software features are too involved to be implemented in a four-week course project. Some projects, such as those involving machine learning, are not really related to the course material. Also, in such a short timeframe, a student does not have time to do an experiment, get results, and write a paper, even if there are other co-authors. So longer projects are needed.

For projects that are related to the course material, having students in a course and seeing how they do on course projects is an ideal way to vet applicants for a longer project. Conversely, for the students, participating in course projects is an excellent way for them to see what the work entails, and what they might be interested in continuing.

Among the longer projects, thesis projects are ideal. Working on a masters thesis gives a student the opportunity to spend multiple semesters on a project. The contributions are in larger chunks than a single-semester independent study, which means that the instructor has less of a need to coordinate work between different individuals, and the project team has less work to merge the projects.

Experience has shown that design is critical to an effective project. The student prepares a proposal, which usually involves two or three rounds of review with the faculty member. Often, a timeline is included, although timelines at the proposal stage have proven to be mostly aspirational, and impossible to adhere to in practice. Students and faculty members meet weekly. Minutes of all meetings are kept in a cloud-based document, such as a Google doc. All artifacts, documentation, and reports are linked to this document. Not only does this serve as a good record of progress, it is also available to students in later semesters to get up to speed on what has already been accomplished. The design of code is reviewed frequently, as it is very easy for students to design their portion in a way that is incompatible with the design of the project as a whole.

When more than one student is interested in a project, it is useful to have students work in groups. That allows students to help each other over rough spots, and cuts the time that the instructor spends in meetings. Whenever possible, it is helpful to let a Ph.D. student direct the group, though the instructor should still exercise oversight of the design and methodology.

Projects fall into several areas. Software development is the most obvious one. New features are added to the peer-assessment system. Tools for managing projects are improved, such as the bots that give feedback to students on the quality of their code. Dev ops is another important kind of project. The project needs a test server to test new features and find bugs. It should be running an up-to-date database. Ideally, new implementations should be rolled out to a portion of the user community so that they can be fully vetted before being deployed to all users.

A second kind of project is mentoring. Students who have done well on their projects in the course are recruited to help students in the next semester. They do a little bit of code development to become familiar with a piece of the project (e.g., the student user interface). Then they participate in writing specifications for student projects. When those projects are assigned to project teams, they meet with those teams weekly, discussing the design, reviewing the code, and overseeing the interface with the rest of the software application.

A third kind of project is to have students develop active-learning exercises for the course [22]. These are not related to the software application per se, but they are very important to the course. Students have developed an average of three or four exercises for each course session. Sometimes they are little programming snippets that have to be finished by the student during class. When they are structured as fill-in-the-blank exercises, they can be autograded by Google forms quizzes during class, allowing students to see when they have found the right answer, and allowing the instructor to display a summary of progress on the monitor in the classroom. We have also adapted industrial training exercises, and development games [23], to be used during class.

The final kind of projects are those related to research. An example is the work mentioned above to use machine learning to recognize a good review. Nearly two dozen students have tried different approaches, such as support vector machines, convolutional neural networks, and bidirectional LSTM. At least half of them have been co-authors on a paper. Others are working on exporting data from the peer-assessment system to facilitate data mining studies or assessment research.

Students are recruited for these projects as follows. About the time that registration starts for the next semester, the instructor prepares a list of possible topics for independent study or thesis research. There are usually 15 to 20 projects in this list. Students are invited to use a Google form to indicate their interest in each project. The instructor looks at the distribution of interest in different projects, and the performance of each student in the course, and makes recommendations to each student on what project might be of mutual interest. A student can accept the recommendation or request a different topic. Of course, many students drop out after indicating interest on the form. The instructor works with the remaining students on proposals for independent study or thesis research. The students register at the beginning of the following semester. Often other students who have not taken the course approach the instructor with interest in one of these projects. They can be assigned projects, such as AI-related projects, that do not rely upon knowing the course material.

2.2 Differentiating roles

Independent-study students can assist TAs, but they should not do the work that the TAs are paid for doing. In particular, they do not need access to student grades, except for the projects that they mentor. In the author's course, projects are graded in a group meeting with TAs and mentors present. The projects are discussed one by one, taking into account the code changes in the repository, the peer reviews done by other students, and the documentation. The mentor for the project (either an independent-study mentor, a TA, or the instructor) is asked to propose a grade for the project. Anyone else can comment on the proposed grade. About half the time, this results in assigning a different grade than originally proposed.

The course has become very successful. It is offered every semester and attracts 150 to 200 students a year, very unusual for a course that is not required for any degree program. It is frequently cited by students as instrumental in helping them land a job. The research that it supports led to a multi-campus NSF grant of nearly $2 million.

3.   Using the ecosystem approach in other courses

The ecosystem approach should be usable in many other courses. The key is looking beyond compartmentalization. Don't ever think that the TA support you are given is the only help you can get with your course. And don't assume that your research assistants are the only students who can help with your research.

Many areas of software engineering and computer science should be amenable to this approach. Directly related to software engineering would be applications for code review and collecting software metrics. If your research involves learning technologies, you might ask students to develop components for Canvas, an open-source LMS. Research on tools for video and class management, like Echo360 [24], would also be relevant. Another example is creating exercises and visualizations for an online textbook, like those written for zyBooks [25]. You might get involved with the Quizi.us project [26] for having students create exam questions through peer

Table 1. Checklist for Constructing a Course Ecosystem

Research

- Can students do projects that will reinforce principles taught in the course *and* benefit the instructor's research?
- Can these projects be divided into pieces, so each student or student team works on a different part of the project?
- Can performance on these projects be used to vet students as candidates for working in the instructor's lab?
- Can data be gathered from student projects or student teams to support experiments in software engineering?
- Can students perform a literature review that will benefit an upcoming paper or research proposal?

Teaching

- Can students to research topics that the instructor might want to cover in a later offering of the course?
- Can students create active-learning exercises over the course material, ideally, for each class session?
- Can student projects be used to enhance the laboratory or software infrastructure?
- Can former students mentor, perhaps for independent-study credit, projects being done by current students?
- Can students write software that will be useful in managing some aspect of the course? A module for an LMS? Adapt a metric to be applied to course projects?

Practice

- In managing student projects, can the instructor practice skills that (s)he can teach to students?
- Can student projects help develop relationships with industry that can lead to joint research or development projects?
- Can the instructor's experience in industry develop internship or funding opportunities for students?

review and machine learning. If your area is intelligent tutoring, you could lead students in working on intelligent tutors for topics that you teach.

Other opportunities are related to research areas that do not involve course tools. One idea is collecting data for educational data-mining experiments, especially when software needs to be modified to enable data collection. If you are teaching computer architecture, you could have students develop simulations for homework, and then offer independent-study projects to enhance the simulation environment. The author spoke with one instructor who has funding from the US Forest Service for work on forest ecology. He recently earned a Ph.D. after a 25-

year career developing software in that field. While he was working on his Ph.D., he had students write software for his research experiments. He has continued this practice, and now has funding from the Forest Service, which has enabled him to hire 30 of his students as interns for about twelve weeks during the summer.

The author talked with an engineering teacher who has implemented this concept in high-school classes. He has students propose physical projects around his lab. They come up with ideas to improve the utilization of tools and equipment. He has them identify a need, for example, developing systems and devices to safely secure tools and machinery when not in use, or organization schemes and structures to improve the lab environment, and then propose a project to deal with it. The students use their knowledge of the engineering design process to plan, design, and execute their projects, and each project concludes with a final professional presentation to the class.

These examples illustrate that courses of many sizes and levels can be structured to incorporate student projects that improve the experience for all students. Table 1 summarizes ideas that may be useful for new engineering faculty. Often, projects can produce research that results in publications or funding. In the case of software projects, designing and managing them helps the instructor keep up his/her skills as a practitioner. There is great synergy to be had from treating a course as an ecosystem of teaching, research, and practice.

Acknowledgments

The author acknowledges the help of Dale Hamilton, from Northwest Nazarene University, and William Wahby, of North Gwinnett High School, who contributed information on how they use this approach in their courses.

References

1. Edward F. Gehringer, Maria Droujkova, Abhishek Gummadi, and Maveeth Nallapeta., “Game mechanics and social networking for co-production of course materials,” Frontiers in Education 2009, paper 2009-1426.
2. Edward F. Gehringer, Abhishek Gummadi, Reejesh Kadanjoth, and Yvonne Marie Andrés, “Motivating effective peer review with extra credit and leaderboards,” ASEE Annual Conference and Exposition, American Society for Engineering Education, Louisville, KY, June 20–23, 2010, paper #2010-1154.
3. Edward Gehringer, Ferry Pramudianto, Abhinav Medhekar, Chandrasekar Rajasekar, and Zhongcan Xiao, "Applications of Artificial Intelligence in Peer Assessment," 2018 ASEE Annual Conference and Exposition, Salt Lake City, June 25, 2018.
4. Sudipto Biswas, Edward F. Gehringer, Dipansha Gupta, Sanket Sahane, and Shriya Sharma, "A Data-Mining Approach to Detecting Plagiarism in Online Exams," EDM 2018:

the 11th International Conference on Educational Data Mining, Buffalo, NY, July 15-18, 2018, pp. 504-507.

5. Edward F. Gehringer, Karishma Navalakha, and Reejesh Kadanjoth, “A student-written wiki textbook supplement for a parallel-architecture course,” Proc. Workshop on Computer Architecture Education, associated with HPCA-17, High-Performance Computer Architecture, San Antonio, TX, Feb. 13, 2011.
6. Edward F. Gehringer, Kai Ma, and Van Duong, "What peer-review systems can learn from online rating sites," PRASAE 2015, 2nd Workshop on Peer Assessment and Self-Assessment in Education, associated with International Conference on Smart Learning Environments, Sinaia, Romania, Sept. 23–25, 2015, in New Horizons in Web-Based Learning, Springer.
7. Ferry Pramudianto, Tarun Chhabra, Edward F. Gehringer and Christopher Maynard, "Assessing the quality of automatic summarization for peer review in education," Second Workshop on Computer-Supported Peer Review in Education, associated with Educational Data Mining 2016, Raleigh, NC, June 29, 2016.
8. Da Young Le, Ferry Pramudianto and Edward F. Gehringer, "Prediction of grades for reviewing with automated peer-review and reputation metrics," Second Workshop on Computer-Supported Peer Review in Education, associated with Educational Data Mining 2016, Raleigh, NC, June 29, 2016.
9. Zhongcan Xiao, Chandrasekar Rajasekar, Ferry Pramudianto, Edward Gehringer, Vishal Chittoor, and Abhinav Medhekar, "Application of neural-network models to labeling educational peer reviews", CSEDM 2018: Educational Data Mining in Computer Science Education Workshop, Buffalo, NY, July 15, 2018
10. Reejesh Kadanjoth, “Workflow Management and Administrative Support for Large Collaborative Projects,” masters thesis, North Carolina State University, July 2010.
11. Ramachandran, Lakshmi, “Automated Assessment of Reviews,” Ph. D. dissertation, North Carolina State University, May 2013.
12. Lakshmi Ramachandran, Edward F. Gehringer, and Ravi Yadav, "Automated assessment of the quality of peer reviews using natural language processing techniques," *International Journal of Artificial Intelligence in Education*, January 2017, pp. 1-48.
13. Yang Song, “Data Sharing in Peer-Assessment Systems for Education,” Ph. D. dissertation,, North Carolina State University, July 2017.
14. Yang Song, Zhewei Hu, and Edward F. Gehringer, "Closing the circle: use of students' responses for peer-assessment rubric improvement," Proc. ICWL 2015, 14th Int'l. Conference on Web-Based Learning, Guangzhou, China, Nov. 5–8, 2015.

15. Yang Song, Zhewei Hu and Yifan Guo, and Ed Gehringer, "An experiment with separate formative and summative rubrics in educational peer assessment", Proc. Frontiers in Education 2016, 46th Annual Conference, Erie, PA, October 12–15, 2016.

16. Yang Song, Zhewei Hu, Edward F. Gehringer, Julia Morris, Jennifer Kidd, and Stacie Ringleb, "Toward better training in peer assessment: Does calibration help?" Second Workshop on Computer-Supported Peer Review in Education, associated with Educational Data Mining 2016, Raleigh, NC, June 29, 2016.

17. Zhewei Hu, “Helping Students Make Better Contributions to Open-Source Software Projects,” Ph. D. dissertation,, North Carolina State University, May 2019.

18. Zhewei Hu, Yang Song, and Edward F. Gehringer, Open-source software in class: students' common mistakes," Proceedings of the 40th International Conference on Software Engineering: Software Engineering Education and Training, Goteborg, Sweden, May-June 2018.

19. Zhewei Hu, Yang Song, and Edward F. Gehringer, “A Test-Driven Approach to Improving Student Contributions to Open-Source Projects,” Frontiers in Education 2019, 49th Annual Conference, Covington, KY, October 16–19, 2019.

20. Zhewei Hu and Edward F. Gehringer, “Improving Feedback on GitHub Pull Requests: A Bots Approach,” Frontiers in Education 2019, 49th Annual Conference, Covington, KY, October 16–19, 2019.

21. Gabriel Zingle, Balaji Radhakrishnan, Yunkai Xiao, Edward Gehringer, Zhongcan Xiao, Ferry Pramudianto, Gauraang Khurana, and Ayush Arnav, "Detecting suggestions in peer assessments," EDM 2019: 12th International Conference on Educational Data Mining, Montreal, July 2019, pp. 474-479.

22. Edward F. Gehringer and Carolyn S. Miller, “Student-generated active-learning exercises,” Proc. SIGCSE 2009, Fortieth Technical Symposium on Computer Science Education, Chattanooga, Mar. 4-7, 2009, pp. 81–85.

23. AgileSparks, “Agile games and exercises list,” https://www.agilesparks.com/resources/topicsubject-reading-lists/agile-games-and-exercises-list/ (Accessed Feb. 2, 2020)

24. Mark, Kai-Pan, Douglas R. Vogel, and Eva YW Wong. "Developing Learning System Continuance with Teachers and Students: Case Study of the Echo360 Lecture Capturing System." In PACIS, p. 170. 2010.

25. Edgcomb, Alex, Frank Vahid, Roman Lysecky, and Susan Lysecky. "Getting students to earnestly do reading, studying, and homework in an introductory programming class." In Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education, pp. 171-176. 2017.

26. Saarinen, Sam, Shriram Krishnamurthi, Kathi Fisler, and Preston Tunnell Wilson. "Harnessing the Wisdom of the Classes: Classsourcing and Machine Learning for Assessment Instrument Generation." In Proceedings of the 50th ACM Technical Symposium on Computer Science Education, pp. 606-612. 2019.