Why computing students should contribute to open source software projects

Spinellis, D.

DOI
10.1145/3437254

Publication date
2021

Document Version
Final published version

Published in
Communications of the ACM

DOI:10.1145/3437254

Diomidis Spinellis

Viewpoint

Why Computing Students Should Contribute to Open Source Software Projects

Acquiring developer-prized practical skills, knowledge, and experiences.

Learning to program is—for many practical, historical, as well as some vacuous reasons—a rite of passage in probably all computer science, informatics, software engineering, and computer engineering courses. For many decades, this skill would reliably set computing graduates apart from their peers in other disciplines. In this Viewpoint, I argue that in the 21st century programming proficiency on its own is neither representative of the skills that the marketplace requires from computing graduates, nor does it offer the strong vocational qualifications it once did. Accordingly, I propose that computing students should be encouraged to contribute code to open source software projects through their curricular activities. I have been practicing and honing this approach for more than 15 years in a software engineering course where open source contributions are an assessed compulsory requirement.[2] Based on this experience, I explain why the ability to make such contributions is the modern generalization of coding skills acquisition, outline what students can learn from such activities, describe how an open source contribution exercise is embedded in the course, and conclude with practices that have underpinned the assignment's success.

IMAGE BY BAKHTIAR ZEIN

Contributing Is the New Coding

Programming skills nowadays are only a part of what a software developer should know. This is the case for two reasons. First, practices have advanced well beyond the chief programmer/surgeon model popularized by Fred Brooks in the 1970s,[1] to include work on orders of magnitude larger systems, advanced tooling, pervasive process automation, as well as sophisticated teamwork, workflows, and management. Second, industrial best practices have homogenized with those followed by large and successful open source software projects. Businesses have assimilated and contributed many open source development practices. This has made the corresponding knowledge and skills por-

table between volunteer projects and enterprise ones.

Consequently, instruction must move from a course's educational laboratory to an organizational setting. By contributing to open source projects, students acquire *in practice* a formidable range of skills, knowledge, and experiences, allowing them to work productively as modern well-rounded developers rather than as the lone-wolf coders portrayed by Hollywood. The most difficult skills to acquire in a traditional programming assignment are the following social and organizational skills.

- Developing a sense of context: understanding how development work is embedded within a project's scope, mission, team of co-developers, and new forms of leadership;
- Interacting with a project's global and diverse community;
- Negotiating feature requests, requirements, and implementation choices;
- Dealing with communication problems, such as absent responses, which are common in volunteer-run projects;
- Appreciating the software as a product through practices such as issue triaging and release planning; and
- Receiving, discussing, and addressing code review comments.

Corresponding learning outcomes associated with technology range from analysis and evaluation to application and creation, including the following:

- Navigating through a project's assets, such as software code, issues, documentation, and pull requests;
- Evaluating swiftly the product and process quality of software systems or components, as is often required in modern software reuse;
- Configuring, building, running, and debugging third-party code;
- Setting up and running software intensive systems with diverse software and hardware requirements. In the course I run, these have included mobile phones, car electronics, application servers, databases, containers, IoT equipment, and embedded devices;
- Choosing realistic contribution goals. (Initially students tend to wildly overestimate their ability to contribute to a project.) This is a key activity in agile development sprints;
- Reading third-party code to identify where their additions or fixes need to be made;
- Modifying a large third-party system by adding a new feature or fixing a bug;
- Writing tests that demonstrate a contribution is working as expected now and into the future;
  - Working with software systems developed using multiple programming languages and tools; students are often surprised to find out that knowledge of an Integrated Development Environment (IDE) is by no means a passport for contributing to a project;
  - Documenting their work, typically using a declarative markup language, for example Markdown or documentation generator code comments;
  - Following sophisticated configuration management (version control) workflows, such as working on issue branches and rebasing code commits; and
  - Passing pre-commit and continuous integration checks and tests.

Both the social and technical learning outcomes are very relevant in the modern workplace—and they go well beyond the proposed ACM/IEEE curriculum for software engineering programs.[3] In parallel, the course's practices embrace many of the ACM/IEEE curriculum guidelines as cross-cutting concerns. These include: the exercising of personal skills, such as critical judgment, effective communication, and the recognition of one's limitations (Curriculum Guideline 8); developing skills for self-directed learning (CG 9); appreciating the multiple dimensions of software engineering problem solving (CG 10); using appropriate and up-to-date tools (CG 12); having a real-world basis (CG 14); and educating through a variety of teaching and learning approaches (CG 18).

We assess the students' performance based on their open source project work available online, their final written report, and their in-class presentations.

Embedding Open Source Development in a Software Engineering Course

The compulsory open source software contribution assignment is part of a third-year course titled "Software Engineering in Practice." (The course received the Management School's Excellence in Teaching award in 2019.) We teach this course to about 20–50 students each year who follow the "Software and Data Analytics Technologies" specialization offered by the Athens University of Economics and Business Department of Management Science and Technology. The course is also a recommended elective for the university's Department of Informatics.

The course is delivered using a (light) flipped classroom approach[4] and is entirely assessed through coursework. The open source contribution assignment counts for 50% of the course's grade. Students can work alone or in pairs. Pairing aims to help students who may feel insecure on their own, though in such cases the pair must deliver more work than an individual, and the contributions must be made from separate GitHub accounts.

We assess the students' performance based on their open source project work available online (code commits and interactions), their final written report, and their in-class presentations. Three presentations take place approximately on week 4 (describing the selected project), week 8 (outlining the proposed contributions), and week 14 (summarizing the contributions' implementation). Getting a contribution accepted is not a prerequisite for passing the assignment, but it is positively assessed. Other assessed elements include the students' comprehension and docu-

mentation of the project they chose, their contribution's breadth, their implementation's quality, their code's integration with the project, their testing implementation, their collaboration with the project's development team, their oral presentations, the quality of their written report, and their use of the available tooling for activities such as version control, code reviews, issue management, and documentation.

Cheating (by copying a contribution from a project fork, or farming out work) could, in theory, be an issue; it is countered by having students present their work in class, and by knowing that their (public) contributions become part of their work portfolio and may be quizzed by future prospective employers.

The course benefits each year from one or two dedicated teaching assistants who run laboratory sessions on key tools, and are available during office hours to advise the students on difficulties they invariably face. The hard work they put into supporting the course, means that increasing the number of attending students would require a commensurate increase in teaching assistants.

Ensuring Successful Open Source Contributions

Students approach the course and its assignment with trepidation and complete it with jubilation. Ensuring students can make meaningful contributions to an open source project requires balancing their inexperience with the fast-paced sophistication of modern, open source software development.

Throughout the years I have given out the assignment, I have seen that contributing to open source projects has become easier. Projects are becoming more inclusive. Many projects have streamlined on-boarding and mentoring, teams are more diverse (including female leads), a published code of contact is common, responses are typically polite, and Windows builds are often supported (though some students adopt Linux to avoid glitches). Contributing has been simplified thanks to handholding in pull request workflows, widespread adoption of continuous integration, diverse code check bots, friendly code review processes, and the use of draft pull requests to allow incremental reviewing of work in progress.

The open source project environment the students dive into is very far apart from the one they typically experience in traditional academic assignments.

Still, the open source project environment the students dive into, is very far apart from the one they typically experience in traditional academic assignments. Therefore, a small-scale contribution is the only realistic goal. The key to making the course's assignment work, is to have what are, on first sight, very low ambitions for the students' contributions. To an undergraduate student, the barriers to open source contributions are often so high, that getting 20 lines of code integrated into a large project is a worthwhile achievement indeed. The advice we give our students for choosing a project can be summarized as follows.

- Choose a project with several active contributors, so that there is a community to guide you and respond to your questions.
- Choose a relatively popular project (some GitHub stars) demonstrating that it provides useful functionality and is developed in a relatively sound way. You want to avoid an abandoned thesis project uploaded on GitHub.
- Avoid very popular projects, so that your contributions will not get drowned in competition, noise, and bureaucracy. (Despite this, we have had students contributing to blockbuster projects, such as Tensorflow and Visual Studio Code.)
- Verify that you can build and run the project on your computer setup.
- Ensure the project regularly accepts pull requests from outsiders, so that yours will also have a chance.
- Try to contribute a trivial fix as a warm-up exercise and as a way to test your ability to follow the project's workflows.
- Look for project issues marked as "Good first issue," which indicate a project that is open to new contributors. (There are several online lists of projects with such issues.)

We leave the choice of the contribution entirely to the students: they can pick an open task from the project's issue database, or propose their own enhancement or fix. Students also often change tack after interacting with the project's core team. Although their freedom to choose their contribution may appear to make the assignment too easy, we have found that it makes it easy enough so that about half of the student contributions get integrated.

The most common problems faced by the students over their assignment are the inability to build the project (typically due to inexperience and platform incompatibilities) and a lack of communication by the project's team (students get needlessly anxious, thinking that their work must be integrated into the project). On the flip-side, the biggest delight felt by the students is when they find their code integrated into production software used worldwide. Invariably, in the course evaluation students comment favorably on the many practical skills and self-confidence they gain after they complete their open source software contribution assignment.

References

1. Brooks, F.P., Jr. *The Mythical Man-Month.* Addison-Wesley, Boston, MA, 1975, 32.
2. Spinellis, D. Future CS course already here. *Commun. ACM 49*, 8 (Aug. 2006), 13; https://bit.ly/3bYxSJs
3. The Joint Task Force on Computing Curricula. Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering. ACM. New York, NY; https://bit.ly/3vn04NP
4. Tucker, B. The flipped classroom. *Education Next 12*, 1 (Mar. 2012), 82–83.

Diomidis Spinellis (dds@aueb.gr) is a professor of software engineering in the Department of Management Science and Technology at the Athens University of Economics and Business, Greece, and a professor of software analytics in the Department of Software Technology at the Delft University of Technology, the Netherlands.

Many thanks to my colleagues Serge Demeyer, Michael Greenberg, and Angeliki Poulymenakou, as well as to the former course students Zoe Kotti and Christos Pappas for their detailed and constructive comments on earlier versions of this Viewpoint.