Creating an Open-Source Ecosystem for Contextualized Learning in Software Engineering

Robert Chatley, Ivan Procaccini, Jason Bailey, Zaki Amin, Estibaliz Fraca
Department of Computing, Imperial College London, United Kingdom.

How to cite: Chatley, R.; Procaccini, I.; Bailey, J.; Amin, Z.; Fraca, E. (2025). Creating an Open-Source Ecosystem for Contextualized Learning in Software Engineering. In: 11th International Conference on Higher Education Advances (HEAd'25). Valencia, 17-20 June 2025. https://doi.org/10.4995/HEAd25.2025.20104

Abstract
Software engineering education is often detached from real-world practices: demonstrations and exercises are limited in scope and fail to capture the complexities and architectures found in industrial projects. Smaller tasks might not demand effective software design so students may neglect these principles in projects.

We present a contextualized learning approach to software engineering education: within our Computer Science department, we develop bespoke software to support teaching and learning with source code made open to student contributions, following open-source software practices. This philosophy enables the use of our code in lectures as real-world examples and allows students to directly shape the evolution of software they use daily, both through occasional contributions and broader internship-like summer projects. These initiatives combine theory and practice so students can experience software evolution, maintenance, and collaboration first-hand in a meaningful context. Our experience could assist other institutions seeking to provide authentic, practical experiences in software engineering.

Keywords: *Contextualized learning; software engineering; open-source; real-world experiences; professional skills.*

1. Introduction

Students enrolling on a specialized computer science or software engineering program aim to acquire industry-relevant skills and knowledge to advance their careers in the technology sector (Kori & Luik, 2020). One problem in teaching these topics effectively is that many aspects of software engineering are only visible in large scale software applications and disappear if small example systems are used (van Deursen et al, 2017). Notably, many of the challenges in software engineering stem from evolving and maintaining a system over time: adding new features to meet user needs, or fixing problems, while maintaining correct behavior for existing

This work is licensed under a Creative Commons License CC BY-NC-SA 4.0
Editorial Universitat Politècnica de València

users (Winters, Manschreck, & Wright, 2020). Creating an authentic scenario where students can experience these dynamics is difficult, especially if we teach with synthetic examples that exist purely for courses and assignments.

In this paper, we present our experiences in adopting an educational approach based on *contextualized learning*: we have developed custom departmental software systems with their source code open to our students. Our software suite includes a bespoke Virtual Learning Environment (VLE) which our students use for all materials and assignments. Building our own software has given us three benefits in terms of contextualized education: 1) we can teach students how software works using examples from familiar systems; 2) we continuously evolve and update these systems, modelling practices of a professional software engineering team; 3) we invite students to contribute to the evolution of these systems, particularly by developing features that assist in their studies. We encourage collaboration between students and staff, allowing students to participate meaningfully in a real open-source software ecosystem.

2. Contextualizing Learning for Software Engineering

Computer Science students typically study both theoretical and practical subjects; some of the most fundamental technical skills to learn are programming and systems design. Curricula often teach several programming paradigms supported by specific programming exercises and assignments. However, projects that are small in scope and short in duration do not provide a context that necessitates the use of broader software engineering practices. Maintaining a system over time is what differentiates programming from software engineering (Winters et al., 2020). Only with a longer-term view does the need for "professional" practices such as diligent use of version control systems, peer code review, or automated quality assurance pipelines (Humble & Farley, 2010) become apparent.

Among several learning theories that provide models to understand and explain how people learn, our work is framed in *contextual learning* theory, or *context-based learning* (Abu-Rasheed, H., Weber, C., & Fathi, M., 2023; Perin, 2011), also denoted as *Contextual Teaching and Learning* (CTL) (Hudson & Whilser, 2008). CTL allows teachers to relate subject matter content to real world situations (Berns & Erickson, 2001) and states that learning takes place when students can construct meaning according to the context of their own lived experiences (Berns & Erickson, 2001; Blanchard, 2001). The learning context - the situation in which learning takes place - describes not only the learner's situation, but also the environment and the interaction between them. CTL has been applied in teaching other domains such as Industrial Engineering and Management (Kukliansky & Rozenes, 2015), giving us confidence to apply it in Computer Science. We believe that an appropriate context to build software engineering skills is one involving systems of a complexity representative of those found in industry, in a domain familiar to the students, with real users to service and demands to be met.

While the wide variety of open-source software (see Section 3) available online offers many real-world examples, most of these are related to domains unfamiliar to our students. We instead chose to make our own software systems open-source, thus creating an effective and accessible learning experience for the students in a directly relevant context.

3. Open-Source Software Development

The software industry has widely adopted practices of *open-source* software development (Fitzgerald, 2006). Open-source software is that whose source code is made freely available, together with a license that permits anyone to view, use, modify, and distribute that source code. Open-source software can provide a valuable educational resource by allowing students to see "inside the box" of real software systems to understand their workings (Brown & Wilson, 2011). Exploring open-source projects is valuable for understanding how real software is structured and implemented, but simply browsing their code does not offer the experience of contributing code, maintenance over longer time periods, or managing changes in response to real-world needs. We think it is valuable to give students experience of contributing to an open-source community as part of their education (Spinellis, 2021) and believe that this can help them to develop important professional skills during their studies.

There are millions of open-source software projects (GitHub, 2022). While many are small hobby projects, there are significant and widely used open-source projects such as the Firefox web browser and the Android operating system for mobile devices. A common characteristic of open-source projects, beyond their free availability, is that any software developer in the world can submit a code change to implement a new feature or fix a bug. Changes are normally considered for inclusion through rigorous review processes. In this way, strong communities of maintainers build up around popular projects, and participating in these communities can become a key part of a professional software developer's work.

Each project has a team of trusted core maintainers. These people have access to edit the code directly. For contributions from the wider community, a contributor creates their own copy of the code (a *fork*), modifies their version to add a new feature or fix a problem, and then makes a *pull request* if they would like their changes to be integrated into the main version. These core maintainers act as gatekeepers for approving pull requests, reviewing any proposed changes carefully and often with extra scrutiny if the author is not a regular contributor to the project. The review process typically involves an extended back-and-forth of feedback and amendments until the maintainers are satisfied and happy to accept the change.

This pattern of interaction is standard in open-source communities. A similar style of code review process is also typical within commercial software development teams to ensure consistent style and high-quality, even if the code in that case is proprietary and closed-source

(Capraro & Riehle, 2016). We believe that providing students with an environment in which they can practice this type of collaborative development is highly valuable.

4. Creating an Internal Open-Source Ecosystem

Our Computer Science department has a long history of developing software to support learning and teaching. Although previously more ad-hoc, we took a more structured approach in 2018, establishing an internal professional software development team. This team has a dual mission: to build systems to facilitate departmental work and support teaching of modern software engineering practices. Its software suite includes a VLE, feedback collection systems, project allocators and an online examination platform.

We made a policy decision to make all our new software open-source and accessible to everyone in the department, including students and staff. This decision has several benefits. First, exposing our software to scrutiny from the very students we train as professional engineers motivates our team to uphold the same coding and process standards we teach. This puts us in a position to use our software as a real-world example in class. Second, it gives students the unique opportunity to contribute as developers to systems they already know as users. This not only provides valuable experience for their résumés but also immerses them in an open-source development culture. Some staff initially worried that exposing our systems' inner workings might enable student exploitation, raising concerns about potential disruption or misuse, especially in areas involving assessment. Despite this, we decided that transparency was the best policy, especially given the benefits of student partnership and collaboration.

We welcome individual student contributions throughout the year, each of which goes through the typical process of code review described in Section 3. As students are busy, these tend to be small but useful changes. Where students have a deeper interest that relates to the educational software context, such as working on learning analytics, we can support this as a capstone project, which allows more time for development and research to be carried out. We also enable more structured forms of partnership with students by offering funded internship-like experiences lasting from six to ten weeks over the summer break, where students work embedded in our team, developing larger features. In total, there have been 31 distinct student contributors on record across our whole ecosystem from 2021-2025. Participating in these projects encourages students to engage with professional practices on long-term collaborative projects, complementing the technical skills they acquire in the formal curriculum.

A recent improvement to our VLE arose from informal discussions between students about navigating their long student record pages. One student took the initiative to develop filters affording a simpler view of upcoming assignments or already graded work. The student coded this feature, which the core development team reviewed and integrated. Small improvements like this occur naturally to students as some of the primary users of our platforms and these quality-of-life improvements are fundamental for maintaining useful software tools. Being able to contribute in this way, to the benefit of themselves and their peers, gives students a strong sense of ownership over the software they use and contextualizes their learning.

In the following section, we present experiences and reflections from students who have contributed to these software systems in recent years.

5. Insights from Student Contributors

We offer our students the opportunity of joining our team as interns over the summer. We started with just two students in the first summer and refined the experience year on year, most recently welcoming a cohort of five. The students were involved in the design and development of different systems and platforms (often *mission-critical*) currently in use in the department.

We conducted a qualitative survey with our past interns, with a mix of open-ended and closed questions, and collected 8 responses. In their responses, students reflected on their experience with professional software engineering practices, particularly agile workflow, code review, deployment with cloud services and task management.

Respondents recalled several key moments that had a particular impact on the way they approach software engineering. A second-year student who worked on our core VLE software remarked: *"[the team's] push for small PRs that [they] could review quickly […] set me up well to work on changes in a much more incremental manner vs one massive PR in future projects"*. Directly exposing students to tools that the team uses to ensure rigor and consistency across our codebases also seemed to have a long-term impact on the students' software engineering habits: *"dev tools like commitlint are pretty much a staple of any project I work on now and are one of the first things I set up"*. Another student positively emphasized the degree of autonomy and initiative they experienced during the project, and how this contributed to their sense of ownership over the software: *"it was a great experience to design the app outline as a team and having the freedom to make these design decisions"*.

Some students reflected on how their placements provided a *"first exposure to the public cloud"* and valued all the skills they developed with respect to cloud technologies. By gaining experience with container and virtualization technology, they learned to deploy their work with different resources to real users, often with system architectures consisting of multiple services.

All respondents described their placement experience as integral to their learning within the context of their degrees: "*It really helped my understanding and made me realize that learning these tools is what makes the whole software engineering experience worthwhile"*.

The closed questions were framed within a 5-point Likert ordinal scale. Table 1 shows the responses to these questions.

Table 1. Responses to scaled questions of the survey completed by 8 summer placement students.

Question	Strongly Agree	Agree	Neither	Disagree	Strongly disagree
My experience in the team improved the way I approach software engineering projects	6	2	-	-	-
The software engineering skills I learnt during my time with the team helped me in the following academic year(s)	6	2	-	-	-
I would like to contribute to other open-source projects after this experience	2	4	2	-	-
The experience helped me improve the way I collaborate in a team	3	4	1	-	-
The experience helped me develop my soft skills	4	3	1	-	-
Overall, the experience helped me grow as a professional beyond what is taught in class and labs	6	1	1	-	-

These responses show that students felt working in this context helped them grow as professionals. Notably, students declare the experience has helped them improve their approach to software engineering projects. This aligns with the contextual teaching and learning theory cited in Section 2. However, we are aware of the survey's limitations, especially its small sample size and the time elapsed from the end of the placements. We do not present these results here as a systematic study from which definitive conclusions can be drawn; nonetheless, these preliminary outcomes and anecdotal experiences suggest there is educational value in the approach we followed. This is reflected in the students internalizing our team's software engineering methodology, then applying it of their own accord in subsequent projects.

6. Recommendations for Educators and Institutions

Where educators and institutions have the flexibility to select tools for teaching and learning, we recommend, where possible, prioritizing the use of open-source software. Open-source tools provide rich opportunities for contextual learning: students can explore how the tools they use every day actually work, deepening their understanding of complex software systems in an environment that feels familiar and meaningful. They may even be able to extend these tools themselves and experience contributing to an open-source community.

While we have chosen to build bespoke tools from scratch, we believe that many of the same educational benefits could be achieved by adopting and engaging with third-party open-source software. Open educational tools —such as learning management systems, assignment submission platforms, or testing frameworks— can serve as authentic case studies in class, or as the basis for exercises and assignments. The key is to allow students to investigate the inner workings of real systems in ways that are closely connected to their own lived experiences.

Some institutions may be understandably hesitant to expose the internals of their software to students, however, we encourage a culture of transparency. We do not rely on "security through obscurity" in which a system is protected by hiding details rather than effective security measures. Instead, we develop application logic in accordance with modern engineering practices, keeping confidential configuration and user data isolated and independent. This separation of concerns makes the code generally safe to open and is in itself a good lesson in software engineering. In our case, sensitive credentials —like those for our databases— are stored as secret variables, which are injected into the code at deployment time and are only accessible to core maintainers. For local development, we auto-generate realistic yet fake data, allowing contributors to work freely without the risk of divulging real information.

By choosing transparent, open tools and building systems with openness in mind, educators may not only enhance learning but also model professional practices that prepare students for working with open systems in the real world.

7. Conclusions

Our open-source ecosystem, giving students the opportunity to work on large projects relevant to their learning, demonstrates the value of contextualized learning in software engineering. By making the internals of our departmental software accessible to students, we provide them with a real-world environment where they can see professional practices in action, contribute meaningfully, and develop skills beyond what is covered in coursework.

This initiative bridges the traditional gap existing between software engineering theory and practice. Through individual contributions and structured internships, students experience working with live systems, engage with peer code review, and follow professional quality assurance processes, gaining exposure to professional practices commonly used in industry. While there were initial concerns about transparency and security, our experience shows that an open, collaborative development model fosters accountability, quality, and engagement. Allowing students to contribute to the tools they use daily creates a virtuous cycle where software development becomes both a learning opportunity and a service to the community.

Moving forward, we aim to further refine and expand this approach, integrating more structured forms of engagement. One possibility is to move beyond contribution being an extra-curricular activity and to create an assignment within the curriculum where all students propose and implement a change to our software systems: this would better enable us to assess long-term impact on student learning and career readiness. Overall, we believe that this model could serve as a blueprint for other institutions seeking to provide authentic, hands-on learning experiences in software engineering teaching and learning.

References

Abu-Rasheed, H., Weber, C., & Fathi, M. (2023). Context based learning: a survey of contextual indicators for personalized and adaptive learning recommendations – A pedagogical and technical perspective. *Frontiers in Education, 8*.

Berns, R.G., & Erickson, P.M. (2001). *Contextual teaching and learning: Preparing students for the new economy.* (The Highlight Zone: Research @ Zork No. 5).

Blanchard, A. (2001). *Contextual Teaching and Learning*. Educational Services.

Brown A. & Wilson, G. (Ed.). (2011). *The architecture of open source applications: elegance, evolution, and a few fearless hacks.* Lulu.com. https://aosabook.org/en/

Capraro, M., & Riehle, D. (2016). InnerSource definition, benefits, and challenges. *ACM Computing Surveys*, *49*(4), Article 67.

Fitzgerald, B. (2006). The transformation of open source software. *MIS Quarterly, 30*(3), 587–598. https://doi.org/10.5555/2017296.2017298

GitHub. (2022, December 6). Octoverse 2022: 10 years of Tracking Open Source. GitHub Blog. https://github.blog/news-insights/research/octoverse-2022-10-years-of-tracking-open-source/

Hudson, C.C., & Whisler V.R. (2008). Contextual teaching and learning for practitioners. *Journal of Systemics, Cybernetics and Informatics, 6*(4). pp. 54-58.

Humble, J., & Farley, D. (2010). *Continuous Delivery: Reliable software releases through build, test, and deployment automation.* Addison-Wesley.

Kori, K., & Luik, P. (2020). Upper- and lower-secondary students' motivation to study computer science. In: Kori, K., Laanpere, M. (eds) *Informatics in schools. Engaging learners in computational thinking (ISSEP 2020)*. Lecture Notes in Computer Science, 12518. Springer. https://doi.org/10.1007/978-3-030-63212-0_6

Kukliansky, I., & Rozenes, S. (2015). The contextual learning approach in engineering education. In *1st International Conference on Higher Education Advances (HEAd'15).* Universitat Politècnica de València, Spain. http://dx.doi.org/10.4995/HEAd15.2015.280

Perin, D. (2011). Facilitating student learning through contextualization: A review of evidence. *Community College Review*, *39*(3), 268-295. https://doi.org/10.1177/0091552111416227

Spinellis, D. (2021). Why computing students should contribute to open source software projects. *Communications of the ACM.* https://cacm.acm.org/opinion/why-computing-students-should-contribute-to-open-source-software-projects/

van Deursen, A., Aniche, M., Aué, J., Slag, R., De Jong, M., Nederlof, A., & Bouwers, E. (2017). A collaborative approach to teaching software architecture. In *Proceedings of 2017 ACM SIGCSE Technical Symposium on Computer Science Education (SIGCSE '17) (pp.* 591–596). Association for Computing Machinery. https://doi.org/10.1145/3017680.3017737

Winters, T., Manschreck, T., & Wright, H. (2020). *Software Engineering at Google: Lessons Learned from Programming Over Time*. O'Reilly Media.