Open Source Software Practices in CS2

A Technical Report submitted to the Department of Computer Science

Presented to the Faculty of the School of Engineering and Applied Science
University of Virginia • Charlottesville, Virginia

In Partial Fulfillment of the Requirements for the Degree
Bachelor of Science, School of Engineering

Emma Choi
Fall, 2021

Technical Project Team Member
Lisa Meng

On my honor as a University Student, I have neither given nor received unauthorized aid on this assignment as defined by the Honor Guidelines for Thesis-Related Assignments

Signature  Date 12/12/2021
Emma Choi

Approved _____ Date _____
John R. Hott, Department of Computer Science

Open Source Software Practices in CS2

Emma Choi
sc5mu@virginia.edu
University of Virginia
Charlottesville, Virginia, USA

Lisa Meng
lm5dc@virginia.edu
University of Virginia
Charlottesville, Virginia, USA

John R. Hott
jrhott@virginia.edu
University of Virginia
Charlottesville, Virginia, USA

ABSTRACT

By contributing to open source software (OSS), students can gain professional software development experience and learn about applications of computer science (CS) concepts in pragmatic contexts. However, integrating such projects in classrooms requires substantial logistical planning by instructors as well as adequate programming skills from students. To mitigate these challenges, we propose four model curricula to serve as accessible strategies of integrating practicable learning opportunities in lower-level CS classes. Depending on classroom circumstances, instructors can assign projects that involve student contributions to OSS, custom plug-ins, simulated open source communities, or practical code excerpts. As a result, students will be able to explore the utility of CS and discover an exciting future in computing.

CCS CONCEPTS

• **Social and professional topics** → *Model curricula*; **Computer science education**; • **Software and its engineering** → **Open source model**.

KEYWORDS

Computer Science Education, Open Source Software, Curriculum Development, CS2

ACM Reference Format:
Emma Choi, Lisa Meng, and John R. Hott. 2021. Open Source Software Practices in CS2. In *21st Koli Calling International Conference on Computing Education Research (Koli Calling '21), November 18–21, 2021, Joensuu, Finland.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3488042.3488047

1 INTRODUCTION

Computer science (CS) is a versatile discipline that empowers students to realize the unlimited possibilities of its application in every domain of knowledge. However, those who give up studying CS describe it as unrewarding, lacking real-world utility, and irrelevant to other fields [1]. In order to train highly capable and inspired students, computer science curricula should extend beyond lectures with technical details to integrate practical contexts in lessons. For example, educators can adopt contextual teaching to motivate, challenge, and engage students [3]. By introducing practical applications of theoretical concepts, students will be able to experiment with pragmatic implementations, perhaps in non-CS domains, and find connections to the wider world.

Layman et al. [10] advocate for meaningful assignments with genuine utility or power to improve society. One such implementation is to integrate open source software (OSS) projects in classrooms. Past case studies show that students who contribute to OSS as part of CS courses benefit from unique opportunities to improve technical and soft skills, interact with open source communities, gain confidence as accomplished developers, and build their portfolios [14]. Additionally, those who work on OSS designed with philanthropic goals, called Humanitarian and Free Open Source Software (HFOSS), provide community service while exploring CS principles in real-world applications. According to Hislop et al. [8], such socially relevant programs have the potential to reinvigorate CS departments by attracting students interested in diverse fields.

Unfortunately, incorporating OSS projects in classrooms poses many logistical difficulties for instructors. For instance, they must filter through millions of projects available on code repositories in order to find those that are appropriate for their course curriculum. Moreover, without prior knowledge of software development principles and tools, novice students can be overwhelmed with learning course content as well as OSS development methodologies [7]. Therefore, Postner et al. [15] suggest that software engineering and capstone courses, where students have more programming experience, are better suited to adopt OSS projects.

As an alternative, we propose several models for introducing OSS projects into lower-level CS courses. With our guide, instructors can identify which strategy best suits their classroom according to its size, student experience, and course objectives. We also provide sample course curricula with handpicked OSS assignments and projects. We specifically target introductory data structures and algorithms courses, also known as CS2, to demonstrate that even inexperienced programmers can engage with OSS. Thus, our proposed curricula reinforce foundational concepts through programming assignments. While exploring the open source movement, students will learn what it takes to become effective computer scientists.

2 BACKGROUND

OSS provides valuable insight into professional software development through publicly available source code, documentation, version history, pending issues, and discussions amongst users and contributors. According to Nascimento et al. [12], students must learn theoretical concepts through such practical contexts so that they can later apply their knowledge in the real world. They discovered that students surveyed in software engineering courses perceived OSS projects to be adequate training for industry jobs. While dissecting complex code, reverse engineering with (sometimes poor) documentation, collaborating with other developers,

Koli Calling '21, November 18–21, 2021, Joensuu, Finland

ACM ISBN 978-1-4503-8488-9/21/11.
https://doi.org/10.1145/3488042.3488047

and testing rigorously, students recognized similarities to their internship and post-grad experiences. As a result, they learned to be resourceful and gained a better understanding of skills needed for professional work [12].

Besides professional development, Ellis et al. [5] suggest that humanitarian OSS projects can demonstrate the capability of computer scientists to succeed beyond the purely technical domain and solve socially relevant problems. They describe the Trinity Sahana project, a disaster recovery coordination software, as an opportunity for students to cooperate with faculty members, corporate associates, and developers across the globe. Based on student surveys, Hislop et al. [8] conclude that similar HFOSS programs can kindle enthusiasm and attract more diverse students to CS.

On the other hand, Postner et al. [15] explain that, in order to implement rewarding HFOSS projects, instructors must overcome the difficulty of finding appropriate software, coordinating with open source communities, and learning relevant technology. But, they are also able to gain new insight into industry practices and engage more with students as mentors rather than passive lecturers. Additionally, the researchers emphasize the importance of communities for instructors to share resources and support each other's efforts to reform their CS curriculum. Meneely et al. [11] present one such example, an online platform where users can circulate OSS suitable for CS course projects and promote the use of open source resources in education. Open Hub [1], OSSpal [2], Foss2serve/POSSE [3], and TeachingOpenSource [4] are other websites with a similar goal.

Prior research has primarily explored the implications of OSS projects in higher-level, usually software engineering, courses. The positive outcomes motivated us to expand the discussion to address logistical challenges faced by instructors and benefit a wider range of CS courses. We combine and expand on current practices to design model curricula, alongside sample schedules and deliverables, suitable for introductory courses such as CS2. The following sections illustrate our proposals, which may also be adapted to other courses to integrate contextual learning through open source software projects.

3 MODELS OF INTEGRATION

We describe four strategies of incorporating OSS development principles, practices, and tools in decreasing degree of engagement with open source communities: direct contributions, custom plug-ins, simulated communities, and code snippets. Depending on the number of students, prior experience with software development, and course objectives, choosing the most appropriate model will best ensure classroom success. Additional resources and guidelines are available on our website [5].

3.1 Direct Contributions to OSS

Programs where students contribute to existing open source codebases involve extensive engagement with communities of OSS contributors. By working on software used and maintained by the public, students can connect with other developers, learn about professional software engineering, and gain a sense of achievement from giving back to the community [2].

However, this implementation raises many challenges for instructors and students. First of all, instructors must scour many code repositories in order to find appropriate OSS to reinforce course concepts without requiring too much supplementary knowledge. Smith et al. [16], describe the laborious process of searching through SourceForge [6] and manually inspecting 1000 projects just to narrow down to a list of 16. For their introductory software engineering course, they specifically looked for criteria such as an active community of contributors, interesting application domain, consistent documentation, and modular architecture to ensure that students can stay engaged, receive timely feedback, and discover both good and bad development practices. We also attempted to find software suitable for CS2 using GitHub [7] search filters and various digital archives of beginner-friendly tasks. Unfortunately, we were not successful in locating more than a handful of projects. Even if students work in groups to complete small tickets, there may not be enough for hundreds of incoming students every semester.

Moreover, real-world software often has a large and complex codebase requiring familiarity with specific development tools. According to Pinto et al. [14], students often struggle to work with insufficient technical skills, understand existing code, and contact other developers for help. Although projects with active contributor communities offer resources and mentorship for novices, instructors should not rely on them as outsourced homework. To avoid overwhelming and overworking students, they must serve as liaisons to OSS communities and provide ample guidance on appropriate coding standards and comprehensive testing. With proper support, students will contribute quality work that can be readily integrated with the rest of the software [6].

All things considered, we recommend course projects requiring student contributions to OSS for smaller classes where instructors can cooperate with OSS communities and devote enough time to help every student. Especially if students have prior experience in software development, they can focus more on learning core course concepts.

3.1.1 *Sample Schedule.*

(1) Identification of suitable OSS: Instructors should first devise a list of criteria to guide their search for appropriate projects. Since each student (or group of students) will be working on different tasks, a rubric will ensure consistency in difficulty and applied CS concepts across projects.
(2) Coordination with contributor communities: A sudden influx of student contributions can overwhelm OSS project managers. Instructors and contributors can arrange an efficient workflow for students to follow contribution guidelines and receive timely code reviews. In addition, they can reserve specific tasks or design new assignments tailored for CS2 students.
(3) Analysis of software: Once students have been paired with projects, they should first explore OSS architecture and documentation to learn how CS principles are employed in real-world software. Instructors can assign deliverables in the

[1] https://www.openhub.net/
[2] https://www.opensoftwaretrends.com/
[3] http://foss2serve.org/
[4] http://teachingopensource.org/
[5] https://oss-classroom.github.io/oss-cs-classroom/
[6] https://sourceforge.net/
[7] https://github.com/

Table 1: Comparison of Proposed Curriculum Models

	Direct Contributions	Custom Plug-ins	Simulated Communities	Code Snippets
Suggested Class Size	Small (less than 50 students)	Medium (less than 200 students)	Medium (less than 200 students)	No restriction
Student Experience with Software Development	Proficiency in code literacy and version control systems (ex. Git)	Competency in code literacy and version control systems	Familiarity with version control systems	No restriction
Course Objectives	Hands-on experience with OSS development	Early introduction to real-world software development	Introduction to software development and exploration of creative project ideas	Exposure to real-world code
Suitability for CS2 Courses	Most difficult for novice students and logistically challenging for instructors	Formidable task in code literacy for students, less coordination necessary with OSS communities	Easier for students to try OSS development and collaborate with classmates, easier for instructors to arrange projects	Most flexible option for students and instructors in terms of workload

form of system architecture diagrams and reports on functionalities and features.

(4) Implementation of solutions: Students will evaluate expected behavior and relevant parts of code before implementing possible solutions and writing tests. Instructors should emphasize the importance of coding conventions so that student work can be easily integrated with the rest of the software.
(5) Examination of code reviews: OSS contributors or instructors should provide feedback so that students can reflect on their work and correct mistakes.

3.2 Custom Plug-Ins for OSS

Relying on an external community of contributors as part of course projects can interfere with class schedules if project managers are slow to assign tasks, provide feedback, review pull requests, or merge changes. Instead, Gehringer [6] suggests that students can build extensions to work on top of existing OSS. In doing so, they will learn from software functioning in the real world without complicated, sometimes bureaucratic, development workflows. By working with APIs and documentation, students do not need to analyze the entire codebase. They will stay within the boundary of their technical expertise and the course scope while brainstorming creative ideas for their plug-ins. Unlike the previous model that requires direct contributions to OSS, instructors only need to select a few OSS for their entire class.

However, this curriculum is limited by the number of OSS that support plug-in development. Software suitable for CS2 must be written in course-designated languages (such as Java, C++, or Python), without complex technical dependencies, while pertaining to a subject matter that can inspire students' imaginations. Furthermore, students may face difficulties in setting up their local environment with OSS and testing their programs. Thus, we recommend that instructors thoroughly investigate and try out OSS with existing plug-ins to gauge project difficulty.

This model is a middle ground between direct contributions to OSS and simulations of open source communities which we describe in the next section. We suggest custom plug-in projects for instructors who want to introduce real-world software while bypassing the logistical challenges of working with external communities. Effectively, students will learn about industry development methods in a sheltered educational setting. They will also be free to pursue creative ideas and make mistakes as their software will not impact the work of other developers or live production code.

3.2.1 Sample Schedule.

(1) Identification of suitable OSS: Software that supports custom plug-in development will often provide guidelines and documentation to support developers. Instructors should consider whether it is accessible and manageable for beginner programmers. Alternatively, instructors can provide their own applications tailored for CS2 students.
(2) Specification of student projects: Instructors should define project requirements so that students can demonstrate CS2 concepts with creative plug-ins.
(3) Analysis of software: Students must first investigate selected OSS to determine what kinds of extensions can be built. They should examine how existing plug-ins add additional functionality to the original software.
(4) Development of plug-ins: Students will build applications by extending existing components and constructing new ones. They can provide design documents to discuss how they utilize specific CS2 concepts within their code.
(5) Distribution of projects: Corresponding to the ethos of the open source movement, students can publish their plug-ins under open source licenses. They should also produce user manuals and documentation to help others install and make contributions to their applications.

3.3 Simulated Open Source Communities

Instructors are often reluctant to integrate open source resources into CS curricula due to their hesitancy to overhaul existing courses with ambitious assignments [19]. To avoid previously mentioned drawbacks of student engagement with OSS and contributor communities, not to mention the associated logistical difficulties for instructors, students can develop projects within their class communities. Without the stress and pressure from directly contributing to public software, they will be more comfortable making mistakes and exploring creative ideas. Instructors will have full control over this sheltered classroom environment where they can select project topics that are more manageable for CS2 students. We recommend a class-centric versioning platform, such as GitHub Classroom [8] or local GitLab [9] servers, to support collaboration among students and help instructors organize assignments. Villarrubia and Kim [18] also provide a collaborative open source software system (SOSS), a version control designed for classrooms. Within this microcosm of OSS communities, students can review and contribute to each other's code. Additionally, instructors can restrict access to their system and recycle project ideas in future semesters.

This curriculum involves no interactions with external OSS communities compared to the previous two models. Although students cannot work with seasoned developers and practical software, a simulated learning environment is more approachable for students studying foundations of programming in lower-level CS courses. Through active (visual, intuitive, role-playing, etc.) and collaborative learning, such a program can encourage new students to enter the computing field [18]. Still, the challenges for students to understand OSS development remain. For instance, instructors may deem that teaching CS2 concepts alongside version control systems can be too overwhelming for students.

Therefore, simulations of OSS communities are more applicable for medium to large classes with a focus on software engineering, where students do not have much development experience. We also recommend this model for instructors who want to grant creative freedom to students in brainstorming project ideas and corresponding software architectures.

3.3.1 *Sample Schedule.*

(1) Identification of host platform: Instructors can set up GitHub Classroom, GitLab, or similar platforms to help students share and maintain their projects online. It should be private or restricted from the public so that students can freely explore software engineering in a simulated development community.
(2) Specification of student projects: Instructors should define project requirements so that students can demonstrate CS2 knowledge with creative applications.
(3) Proposal of project outlines: Students will brainstorm project ideas and design an outline of their software. Instructors should guide students during this process in order to verify the feasibility of projects. Students can submit design documents to demonstrate their usage of relevant CS2 concepts.
(4) Development of project: Students will build their own software and write tests. They can also compose documentation for other users and contributors.
(5) Inspection of other student projects: Students can review classmates' software to explore diverse applications of CS principles and practice code literacy skills while investigating the features and structures of other projects.
(6) Contribution to other student projects: Students will identify bugs and brainstorm new features that can be added to improve their classmates' software. They will assume the roles of OSS developers to make contributions and improve each other's work.

3.4 OSS Code Snippets

Our last proposal does not involve programming assignments. Instead, instructors can integrate code snippets from OSS into lectures as real-life examples of textbook concepts. This curriculum is especially beneficial for lower-level courses like CS2 where the primary objective is to introduce fundamental concepts. To aid the search for relevant applications, de França Tonhão et al. [4] provide an online repository [10] with deconstructed software solutions that demonstrate CS concepts. As an alternative, instructors can employ more interactive activities to engage students. For example, they can examine OSS source code and analyze components that illustrate specific topics. Alternatively, instructors can integrate code snippets into programming exercises.

As students are not required to contribute to OSS or build complete applications of their own, they may not gain as much insight into professional software development. Still, they will have the opportunity to study CS concepts in the context of practical applications.

We recommend this model to instructors who want to enrich course content by introducing real-world software without organizing large, long-term projects. All things considered, it is a versatile addition for any course regardless of enrollment size or student experience.

3.4.1 *Sample Schedule.*

(1) Identification of suitable OSS: Instructors should find software that is of appropriate complexity in an application domain interesting to students. They can also encourage students to explore different OSS on code repositories.
(2) Illustration of CS2 topics: Instructors should identify components of selected OSS that can illustrate course materials and demonstrate how they work with the rest of the code.
(3) Evaluation of student understanding: Students can study how provided OSS code snippets utilize classroom concepts and reflect on lessons learned. Depending on specific implementations of this strategy, instructors can assign reports, quizzes, or programming exercises.

4 DISCUSSION AND FUTURE WORK

While presented for CS2 programs, instructors can adapt our proposed models to suit their course objectives and classroom needs. In typical CS2 classes, most students lack sufficient knowledge about

[8] https://classroom.github.com/
[9] https://gitlab.com/
[10] https://portalworkedexamples.000webhostapp.com/

software engineering principles and tools to contribute to existing OSS. Therefore, we recommend introducing code snippets from OSS then gradually integrating more interactive assignments and projects.

Lessons involving OSS often call for some degree of knowledge in software development. Furthermore, contributing to projects with real-world impacts may invoke reluctance and apprehension in some students [13]. Thus, instructors may worry that CS2 courses are too early to introduce software engineering principles, where students are just starting to learn how to write basic programs. However, most CS majors study collaborative software development at some point in their education, so introducing those concepts as part of CS2 can establish a solid foundation for future courses and train students to follow good coding practices. For example, research shows that students become more confident with unfamiliar tools, such as version control systems, with more exposure, and the additional challenge does not negatively impact academic performance [17] [9].

Just as much as students learn from participating in OSS projects, open source contributor communities should also benefit from their contributions. Therefore, instructors must help students produce quality work that will meet the standards of OSS project managers. Following students' progress from examining source code to brainstorming and implementing solutions to code reviews, instructors should promote good engineering and programming principles. They can duly support students by providing informative resources, clear instructions, and patient guidance.

Context-based teaching with OSS projects highlights diverse and meaningful pursuits of computer scientists. By exploring applications of technical concepts in the real world, students can expand their perspectives and discover versatile utilities of CS concepts. As a result, this curriculum can better engage students and broaden participation in the field. Moreover, instructors can provide valuable training for internships and industry jobs by introducing lessons from real-world artifacts. Students can also showcase impressive open source projects online as part of their portfolios. All of the above benefits can be readily acquired by adopting one of our four strategies.

Each model of integrating OSS in CS2 classrooms poses unique difficulties and drawbacks. For future research, we hope to apply and compare each model in a foundational data structures and algorithms course. By gathering instructor and student feedback, we will be able to address new challenges and further improve our proposed strategies.

5 CONCLUSION

We present four models of integrating OSS projects, principles, and practices into introductory CS courses with varying degrees of engagement with real-world software and open source communities. Depending on classroom needs, instructors will be able to identify the most suitable strategy to teach students theoretical concepts in the context of realistic problems. We hope that our suggested plans of action can facilitate discussions regarding the reformation of CS curricula to better engage, inspire, and prepare the future generation of computer scientists capable of solving socio-technical problems.

REFERENCES

[1] Maureen Biggers, Anne Brauer, and Tuba Yilmaz. 2008. Student Perceptions of Computer Science: A Retention Study Comparing Graduating Seniors with Cs Leavers. *SIGCSE Bull.* 40, 1 (March 2008), 402–406. https://doi.org/10.1145/1352322.1352274

[2] Grant Braught, John Maccormick, James Bowring, Quinn Burke, Barbara Cutler, David Goldschmidt, Mukkai Krishnamoorthy, Wesley Turner, Steven Huss-Lederman, Bonnie Mackellar, and Allen Tucker. 2018. A Multi-Institutional Perspective on H/FOSS Projects in the Computing Curriculum. *ACM Trans. Comput. Educ.* 18, 2, Article 7 (July 2018), 31 pages. https://doi.org/10.1145/3145476

[3] Steve Cooper and Steve Cunningham. 2010. Teaching Computer Science in Context. *ACM Inroads* 1, 1 (March 2010), 5–8. https://doi.org/10.1145/1721933.1721934

[4] Simone de França Tonhão, Thelma Elita Colanzi, and Igor Steinmacher. 2020. A Portal for Cataloging Worked Examples Extracted from Open Source Software. In *Proceedings of the 34th Brazilian Symposium on Software Engineering* (Natal, Brazil) *(SBES '20)*. Association for Computing Machinery, New York, NY, USA, 493–498. https://doi.org/10.1145/3422392.3422471

[5] Heidi J. C. Ellis, Ralph A. Morelli, Trishan R. de Lanerolle, Jonathan Damon, and Jonathan Raye. 2007. Can Humanitarian Open-Source Software Development Draw New Students to CS? *SIGCSE Bull.* 39, 1 (March 2007), 551–555. https://doi.org/10.1145/1227504.1227495

[6] E. F. Gehringer. 2011. From the manager's perspective: Classroom contributions to open-source projects. In *2011 Frontiers in Education Conference (FIE)*. F1E–1–F1E–5. https://doi.org/10.1109/FIE.2011.6143028

[7] Swapna S. Gokhale, Thérèse Smith, and Robert McCartney. 2012. Integrating Open Source Software into software engineering curriculum: Challenges in selecting projects. In *2012 First International Workshop on Software Engineering Education Based on Real-World Experiences (EduRex)*. IEEE, 9–12. https://doi.org/10.1109/edurex.2012.6225697

[8] Gregory W. Hislop, Heidi J.C. Ellis, and Ralph A. Morelli. 2009. Evaluating Student Experiences in Developing Software for Humanity. *SIGCSE Bull.* 41, 3 (July 2009), 263–267. https://doi.org/10.1145/1595496.1562959

[9] Jonas Isacson and Emil Lindblom. 2017. Correlation of User Behaviour Patterns and Assignment Supplements in KTH-GitHub Repositories.

[10] Lucas Layman, Laurie Williams, and Kelli Slaten. 2007. Note to Self: Make Assignments Meaningful. *SIGCSE Bull.* 39, 1 (March 2007), 459–463. https://doi.org/10.1145/1227504.1227466

[11] Andrew Meneely, Laurie Williams, and Edward F. Gehringer. 2008. ROSE: A Repository of Education-Friendly Open-Source Projects. In *Proceedings of the 13th annual conference on Innovation and technology in computer science education - ITiCSE '08 (ITiCSE '08)*. ACM Press, 7–11. https://doi.org/10.1145/1384271.1384276

[12] Debora M. C. Nascimento, Christina F. G. Chavez, and Roberto A. Bittencourt. 2018. The Adoption of Open Source Projects in Engineering Education: A Real Software Development Experience. In *2018 IEEE Frontiers in Education Conference (FIE)*. IEEE, 1–9. https://doi.org/10.1109/FIE.2018.8658908

[13] Michela Pedroni, Till Bay, Manuel Oriol, and Andreas Pedroni. 2007. Open Source Projects in Programming Courses. *SIGCSE Bull.* 39, 1 (March 2007), 454–458. https://doi.org/10.1145/1227504.1227465

[14] Gustavo Pinto, Clarice Ferreira, Cleice Souza, Igor Steinmacher, and Paulo Meirelles. 2019. Training Software Engineers Using Open-Source Software: The Students' Perspective. In *Proceedings of the 41st International Conference on Software Engineering: Software Engineering Education and Training* (Montreal, Quebec, Canada) *(ICSE-SEET '19)*. IEEE Press, 147–157. https://doi.org/10.1109/ICSE-SEET.2019.00024

[15] Lori Postner, Darci Burdge, Heidi J. C. Ellis, Stoney Jackson, and Gregory W. Hislop. 2019. Impact of HFOSS on Education on Instructors. In *Proceedings of the 2019 ACM Conference on Innovation and Technology in Computer Science Education* (Aberdeen, Scotland Uk) *(ITiCSE '19)*. Association for Computing Machinery, New York, NY, USA, 285–291. https://doi.org/10.1145/3304221.3319765

[16] Therese Mary Smith, Robert McCartney, Swapna S. Gokhale, and Lisa C. Kaczmarczyk. 2014. Selecting Open Source Software Projects to Teach Software Engineering. In *Proceedings of the 45th ACM Technical Symposium on Computer Science Education* (Atlanta, Georgia, USA) *(SIGCSE '14)*. Association for Computing Machinery, New York, NY, USA, 397–402. https://doi.org/10.1145/2538862.2538932

[17] Gina Sprint and Jason Conci. 2019. Mining GitHub Classroom Commit Behavior in Elective and Introductory Computer Science Courses. *J. Comput. Sci. Coll.* 35, 1 (Oct. 2019), 76–84.

[18] Andrew Villarrubia and Hyunju Kim. 2015. Building a community system to teach collaborative software development. In *2015 10th International Conference on Computer Science Education (ICCSE)*. 829–833. https://doi.org/10.1109/ICCSE.2015.7250360

[19] Linh N Vu, Ti Kean Tan, and Prapat Maneerat. 2004. Incorporating open-source software development into computer science and software engineering education at university level. In *Proceedings of the Second Australian Undergraduate Students' Computing Conference*. Citeseer, 149–164.