Scaffolded Projects for the Social Good: A Strategy for Deploying Studio Model in CS Education

Stan Kurkovsky
kurkovsky@ccsu.edu
Central Connecticut State University
USA

Mikey Goldweber
mikeyg@denison.edu
Denison University
USA

Nathan Sommer
sommern1@xavier.edu
Xavier University
USA

Chad A. Williams
cwilliams@ccsu.edu
Central Connecticut State University
USA

ABSTRACT

Scaffolded Projects for the Social Good (SPSG) is an adaptable service-learning framework with a low adoption threshold based on the studio model. Its goal is to enable instructors to easily embed externally sourced projects supporting Computing for the Social Good (CSG) concepts into existing software engineering or similar courses and addresses the barriers common to service learning, as well as other frameworks with similar CSG-related objectives. Establishing connections between computing and its societal benefits has proven to be an effective strategy for attracting students, especially those from underrepresented groups within the discipline. Furthermore, this work supports competency-based learning by offering students an opportunity to solve real-world problems in an authentic environment using current industrial practices and tools coupled with strong mentoring support from volunteer professionals and near-peers. Using a studio model helps overcome the timing impedance between the length of a single academic term and the timeframes required to complete real-world projects with student teams.

ACM Reference Format:
Stan Kurkovsky, Mikey Goldweber, Nathan Sommer, and Chad A. Williams. 2024. Scaffolded Projects for the Social Good: A Strategy for Deploying Studio Model in CS Education. In *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 2 (SIGCSE 2024), March 20–23, 2024, Portland, OR, USA.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3626253.3635487

1 THE SPSG FRAMEWORK

Rationale: The concept of a software studio is based on the idea that a CS degree does not prepare a graduate to be a great software engineer any more than a degree in Arts prepares one to be a great artist. However, repeated practice under the guidance of experienced mentors within a studio framework can help develop student competencies to a significantly higher level [1]. The concept of a software studio has been successfully applied at both graduate [3, 4] and undergraduate levels [5, 6]. Current literature suggests that participating in software studio projects offers multiple advantages to students. These include fostering a heightened sense of ownership, which arises from the extended project duration and real-world application. Student learning is improved through mentoring by professionals, and students become more adept at using professional practices and tools. Additionally, these projects improve project outcomes through customer involvement and support reflective practices through retrospectives and peer feedback.

Implementing a studio-based approach to software projects is challenging. Setting up a logistical infrastructure requires a significant investment of faculty effort to create a project workflow that would fit their course or program needs. One of the primary challenges is that students participate in the studio projects for fixed periods of time (1, 2, or even 3 terms), while projects exist in the studio as long as needed. While some large institutions can provide staff and other resources to support such projects, this may be unrealistic for many teaching-focused institutions. There must be an established procedure for evaluating the project scope, which often relies on an internalized process involving program faculty, staff, and well-established partners sponsoring the projects. Each institution's studio approach is tightly integrated with the program curriculum and the institutional context, which could make this experience very difficult to adopt elsewhere.

The authors have accumulated a broad spectrum of experience working with various studio-based approaches for a wide variety of partners in over 75 projects. This experience has illuminated the challenges involved in such projects such as the difficulties of scaling the approach to large class sizes, identifying projects of an appropriate size, addressing project maintenance, and coordinating project handoff. Over time, these projects helped form a highly scaffolded framework that provided enough structure to ensure project success while also being flexible, so teams had sufficient agility for adapting to evolving requirements or emerging challenges.

Work Done: The authors are currently developing an easy-to-adopt framework to deploy studio-based projects, which is based on agile principles and the lessons learned from the HFOSS initiative [2]. It will provide instructors with a customizable set of curricular materials aimed to streamline the software project workflow and offer ample mechanisms for formative and summative assessment of student learning. Students participating in SPSG projects will gain

SIGCSE 2024, March 20–23, 2024, Portland, OR, USA
© 2024 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-0424-6/24/03
https://doi.org/10.1145/3626253.3635487

the experience of applying their knowledge and skills in an authentic project environment, preparing them to tackle the challenges of real-world projects. SPSG will help students make meaningful contributions to the missions of non-profit and community organizations while providing concrete examples regarding the impact of computing on society. SPSG will also make it easier for instructors to engage with external software project partners by providing a project feasibility assessment model.

Our current implementation of the SPSG framework builds on top of one or two upper-level courses commonly found in many programs: Software Engineering (SE) and Senior Project (SP). The SE course typically combines software engineering fundamentals with a semester-long course project. The SP course focuses primarily on the project and is often supplemented by readings and discussions emphasizing professional and ethical issues. A typical SPSG studio project can last 2 to 3 semesters. Students in both SE and SP courses can form teams working on a variety of projects for different customers. After the first semester, students who worked on a given project in SE, continue into SP and work with new SE student teams ensuring project continuity and knowledge transfer. Other options for project continuity are described below.

The Framework: Our framework uses a studio-based project organization that is grounded in agile practices by focusing on short iterations, partner (customer) involvement, and providing enough flexibility to address changing requirements and other challenges. Each course project consists of four phases: inception, elaboration, development, and transition. The inception phase occurs before the project commences and involves several key elements. During this phase, the instructor collaborates with the project partner to assess project feasibility, align it with the capabilities of student teams, define project scope and duration, and establish expected outcomes. This results in a standardized project proposal document, which is then received by one or more student teams.

During elaboration, students (or the instructor) form teams and work closely with the project sponsor to understand and formulate project requirements. For new projects, teams go through several iterations of project partner interviews to form outline requirements, which then are translated into user stories forming the initial product backlog. For continuing projects, remaining user stories are reviewed and adjusted taking into account any new user stories reflecting the next set of requirements. Teams work closely with the project partner to verify that the user stories correctly reflect the desired functionality, while the partner sets user story priorities.

The development phase is structured using the agile methodology. Each two-week sprint starts with the team planning their work by identifying which user stories will be moved from the product backlog into the sprint backlog. Given that a student might dedicate no more than 10 hours per week to the project, instead of daily scrum meetings, students participate in a weekly scrum during one of the regularly scheduled class meetings. During a weekly scrum, each team briefly discusses their progress made during the past week towards meeting the sprint goal, as well as the plan for the upcoming week. Throughout the sprint, teams maintain an open line of communication with the project partner to make sure that any questions can be answered promptly. At the end of the sprint, each team demonstrates their progress to their project partner during a sprint review when they may accept or reject some or all of the newly developed functionality. Each team also participates in an in-class self-reflection during the sprint retrospectives when they can inspect the results they accomplished, discuss the lessons learned during the sprint, and adapt both the product backlog (to better reflect any possible changes in the project requirements) and the team's workflow (to make sure that the team does more of what works for them and less of what doesn't).

The transition phase takes place during the last week of the semester. It focuses on knowledge transfer and typically includes the final project demonstration to the project partner and the entire class. There are several possibilities at this point: a) the project continues next semester with the same team(s) (typical for teams in the SE course that will be in the SP course next semester); b) the project continues next semester with at least one new team (same scenario as above, but with an additional team from the next semester's SE course; or when there's no overlap in teams); or c) the project is completed and there are no teams actively involved in it (aside from routine maintenance).

Regardless of the project state, to facilitate the knowledge transfer, teams typically prepare a set of user and deployment documents for the project partner and the team(s) that may continue working on the same project. Project continuity is enabled by several factors: a) team staggering so that there is an overlap between a team continuing with the project and a new team, b) extensive knowledge transfer documentation including configuration and build documents, and c) near-peer mentoring where a recent graduate who worked on the same project mentors the next team(s).

Throughout their work on the projects, student teams are typically required to produce a range of deliverables in the form of reports, presentations, source code commits, project demonstrations, meeting notes, etc. Each of these deliverables is an opportunity for formative assessment, while those produced towards the end of the project and especially at the project transition phase are very useful for summative assessment of student learning.

2 ACKNOWLEDGEMENTS

This work was supported in part by NSF awards 2315322 and 2315323.

REFERENCES

[1] Christopher N. Bull and Jon Whittle. 2014. Supporting Reflective Practice in Software Engineering Education through a Studio-Based Approach. *IEEE Software* 31, 4 (2014), 44–50. https://doi.org/10.1109/MS.2014.52

[2] Heidi J. C. Ellis, Gregory W. Hislop, Stoney Jackson, and Lori Postner. 2015. Team Project Experiences in Humanitarian Free and Open Source Software (HFOSS). *ACM Trans. Comput. Educ.* 15, 4, Article 18 (dec 2015), 23 pages. https://doi.org/10.1145/2684812

[3] Phillip A. Laplante. 2006. An Agile, Graduate, Software Studio Course. *IEEE Transactions on Education* 49, 4 (2006), 417–419. https://doi.org/10.1109/TE.2006.879790

[4] David Root, Mel Rosso-Llopart, and Gil Taran. 2008. Exporting Studio: Critical Issues to Successfully Adopt the Software Studio Concept. In *2008 21st Conference on Software Engineering Education and Training*. 41–48. https://doi.org/10.1109/CSEET.2008.21

[5] Daniela Rosca. 2018. Acquiring Professional Software Engineering Skills through Studio-based Learning. In *2018 17th International Conference on Information Technology Based Higher Education and Training (ITHET)*. 1–6. https://doi.org/10.1109/ITHET.2018.8424773

[6] Robbie Simpson and Tim Storer. 2017. Experimenting with Realism in Software Engineering Team Projects: An Experience Report. In *2017 IEEE 30th Conference on Software Engineering Education and Training (CSEE&T)*. 87–96. https://doi.org/10.1109/CSEET.2017.23