Navigating Multi-Semester Service Learning in Software Engineering: Strategies for Success

Chad A. Williams
Stan Kurkovsky
Department of Computer Science
Central Connecticut State University
New Britain, CT, USA
{cwilliams, kurkovsky}@ccsu.edu

Mikey Goldweber
Department of Computer Science
Denison University
Granville, OH, USA
goldweberm@denison.edu

Nathan Sommer
Department of Computer Science
Xavier University
Cincinnati, OH, USA
sommern1@xavier.edu

2025 IEEE Frontiers in Education Conference (FIE) | 979-8-3315-0105-1/25/$31.00 ©2025 IEEE | DOI: 10.1109/FIE63693.2025.11328693

***Abstract*—This innovative practice full paper examines strategies for managing multi-semester service learning projects in software engineering courses. Project-based learning is a cornerstone of software engineering education that engages students with real-world challenges through hands-on experiences. While these projects provide valuable experiential learning opportunities and expose students to complex, real-world challenges, they also present logistical difficulties, particularly in sustaining momentum and ensuring effective knowledge transfer between student teams. Grounded in existing research, this work presents a flexible framework of strategies and best practices developed by the authors over the last 10 years across various institutional contexts. The paper evaluates the effectiveness of these strategies through continuous assessment mechanisms, including iterative feedback and student reflections. Findings contribute actionable methodologies for integrating multi-semester projects into curricula, supporting sustainable service learning, and better preparing students for professional software development practice.**

***Index Terms*—Service learning, Project-based learning, Computer science, Software Engineering**

I. Introduction

Project-based learning is a cornerstone of software engineering education that fosters critical thinking, problem-solving, communication, and teamwork skills through hands-on engagement with real-world challenges [1].While industry partnerships offer structured projects, scaling these projects and managing handovers can be difficult without in-house technical support [2].

Service learning offers an alternative by engaging students with local non-profits, but these projects often require a multi-semester approach to address maintenance due to partners often lacking in-house software engineering expertise [3]. This can lead to unfinished projects or an undue burden on instructors to maintain them beyond a single semester. To ensure a sustainable lasting impact, a successful strategy for managing multi-semester service-learning projects is essential [4]. This paper presents a flexible framework of strategies and best practices, drawing on over 10 years of experience running multi-semester software engineering projects using a variety of models. Recognizing that no two institutions are identical, this framework enables educators to select and adapt strategies to ensure project success.

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

II. Prior Work

The authors have broad experience managing multi-semester student software projects across various institutions. At Bemidji State University, a small public college in Minnesota, Williams led a year-long project where multiple student teams in Software Engineering I and II courses collaborated on a single project for a local Boys & Girls Club. At Xavier University, a small private liberal arts college in Ohio, Goldweber devised an initiative [5] for service-learning projects to persist across semesters, ensuring the best match between project needs, student team capabilities, and the involved courses. Goldweber also led a two-semester effort to develop mobile apps to assist with their city's opioid addiction crisis. Since 2014, Kurkovsky has been running a Software Engineering Studio (The Studio) at Central Connecticut State University, a large public school. The Studio has connected external customers with teams of undergraduate students working on software development projects, involving over 160 teams consisting of 4-5 students in over 75 unique projects, many of which were for non-profit or community organizations. Over the years, these projects have helped form a highly scaffolded framework that provides enough structure to ensure project success while also being flexible, allowing teams to adapt to evolving requirements or emerging challenges. Additionally, Williams has led projects within The Studio where teams continued work initiated by different instructors and transitioned their work to other instructors, ensuring continuity and effective knowledge transfer. At The College of Wooster, a small private liberal arts school, Sommer has advised over a dozen student teams as part of an 8-week summer program where teams of 2-4 students work for clients, both commercial and non-profit, and is now adopting the Scaffolded Projects for the Social Good (SPSG) framework in Software Engineering II and senior capstone courses at Xavier University.

Each author has led multi-semester projects for multiple years and found similarities both in challenges and in effective practices. Two decades ago, some of us began with single-semester, single-team projects. To accommodate larger, more

realistic projects and enhance learning, we independently gravitated to a studio-based approach, removing constraints on project size or duration. Early attempts, like floating projects between semesters, revealed crucial insights into handover limitations and the need for universal faculty buy-in.

A software studio [6] is an environment where a succession of student teams work on projects that can span multiple semesters, facilitating knowledge transfer and mentoring. This model holds strong potential for building student competencies [7] and bridging the gap between computing education and industry needs [8]. While challenges in external projects often focus on technical problems [9] or integration into their curriculum [10], significant difficulties arise from misaligned goals between partners and students [11], partners' limited understanding of software development [12], and academic constraints [13]. Many project partners do not understand what it takes to create working software of any appreciable complexity [12]. Some of these challenges can be minimized by involving alumni in helping source projects [11] and by emphasizing (or even enforcing) frequent interactions with external partners [10]. Humanitarian Free and Open Source Software (HFOSS) projects present similar challenges, particularly the absence of a distinct external project partner for scope discussions [14], [15].

Service learning is a pedagogical approach integrating community service with instruction and reflection, enhancing learning, civic responsibility, and community engagement [16]. It has been shown to increase student interest in computing careers, particularly among underrepresented groups [17], [18], and improve abilities to apply knowledge [19], persistence [20], motivation [21], self-efficacy [22], leadership [19], and civic duty [23]. While highlighting these benefits, a systematic literature review by Salam et al [24] highlights common challenges that include logistics (communication, time management) and academic aspects (assessment, connection between student and project outcomes).

A. Challenges of multi-semester efforts

Multi-semester service-learning projects offer significant opportunities for students to engage with complex, real-world challenges, but they also present several logistical difficulties. Sustaining momentum and ensuring effective knowledge transfer between student teams is a primary challenge [25]. As projects span multiple semesters, it becomes crucial to maintain continuity in development practices and documentation protocols. Without robust documentation and organized code repositories, incoming teams may struggle to understand the project's current state, leading to delays. Mentorship systems where experienced students or alumni guide new teams can enhance knowledge transfer, but coordinating these efforts proves challenging [9].

Another major challenge is managing the scope of multi-semester projects. Projects often evolve rapidly, and the initial requirements may change as the project progresses. This can lead to difficulties in prioritizing features and adapting to new requirements. Regular collaboration with project partners to revise scope is essential to ensure alignment with current priorities and completed work. Furthermore, engaging with external partners requires frequent interactions to align goals and expectations, but partners may not always understand academic constraints, especially concerning project cadence and timeframes [26]. These challenges highlight the need for flexible frameworks that can adapt to different institutional contexts and provide continuous improvement practices to enhance project outcomes.

B. Motivation and Context

This work is motivated by the need to enhance software engineering education through experiential and project-based learning. Multi-semester service-learning projects offer significant opportunities for students to engage with complex, real-world challenges, fostering critical thinking, problem-solving skills, and teamwork [21]. However, these projects also present logistical difficulties, particularly in sustaining momentum and ensuring effective knowledge transfer between student teams continuing each other's work [27]. By addressing these challenges, we hope to provide a framework that enhances the educational impact and scalability of multi-semester service-learning initiatives.

Integrating the development of professional skills like teamwork and communication in software engineering education is crucial [28]. Embedding service-learning projects into the curriculum provides valuable hands-on experience, allowing students to apply theoretical knowledge to solving practical problems in an authentic context. This approach has been shown to enhance student motivation, retention, and overall academic performance, creating a dynamic and engaging learning environment. However, managing multi-semester projects requires structured methodologies to overcome numerous challenges, including effective knowledge transfer and scope management [9]. Addressing these issues with a flexible framework helps prepare students to meet the challenges of professional software development.

III. Practice and Evolution

A. Software Engineering Studio Model

The Studio model at Central Connecticut State University has been a cornerstone of our approach to managing multi-semester service-learning projects. Since 2014, The Studio has connected external project partners with student teams, typically comprised of 4-5 seniors, working on software engineering projects spanning one to four semesters. Over the years, The Studio has collaborated with many distinct project partners, engaging more than 650 students in various projects, many for non-profit or community organizations.

The Studio's workflow has evolved into the Scaffolded Projects for the Social Good (SPSG) framework, that aligns software development projects with student learning outcomes and course objectives. This framework ensures project success while providing flexibility for teams to adapt to evolving requirements and emerging challenges. SPSG emphasizes the importance of robust documentation protocols, organized code

repositories, and mentorship systems to facilitate effective knowledge transfer and sustained development practices.

Projects facilitated by the Studio are embedded into several software engineering courses, where student teams participate in all phases of the software development lifecycle under the guidance of the course instructor and in close cooperation with the project partner. Formative and summative assessment of student work is facilitated by a broad range of project-related deliverables spread throughout the academic term, enabling the course instructor to provide timely feedback and ensure that students meet their learning outcomes. Regular interaction with the project partner helps ensure that each team makes good progress towards meeting project requirements.

The Studio framework includes four phases: *inception*, *elaboration*, *development*, and *transition*. During the inception phase, the course instructor negotiates the project scope and other considerations with the project partner. In the elaboration phase, teams form, write their working agreements, and undertake requirements engineering. The development phase involves teams going through four to six two-week development sprints. Finally, in the transition phase, teams prepare for knowledge transfer and either deliver their work to the project partner or transfer it to the next team(s). This structured approach ensures that the project scope aligns well with student team capabilities, technical project requirements, and academic term constraints [29], [30]. The integration of the practices discussed in this work across the project lifecycle is visually summarized in Figure 1, which maps each SPSG practice to the inception, elaboration, development, and transition phases where it is most applicable.

The Studio has proven successful with a broad range of partners, including commercial, industrial, non-profit, and community organizations. By providing a flexible framework that supports continuous improvement and effective project transitions, SPSG's framework's implementation of the studio model enhances the scalability and sustainability of service-learning projects. This approach prepares students for the complexities of professional software development while making a meaningful impact on the community. Our ongoing efforts using this framework to support software projects for the common good are documented in [31].

B. Evolution of Multi-Semester Projects

Central Connecticut State University's (CCSU) Software Engineering Studio started out in 2014 with a collaboration with one of our alumni who graduated in the mid-2000's. After over a decade of professional software engineering and leadership experience with major technology and financial companies, they founded and led several technology startups. They had an idea for a project with the potential to become a startup company, but it first needed a proof-of-concept to test the feasibility and practicality of the resulting solution.

Around the same time, our curriculum was changing, and we added the existing Senior Project course (which was previously an elective) to the required capstone sequence for our ABET-accredited computer science program. At the time, the first course in the sequence, Software Engineering, required students to come up with their own project ideas, but we decided that going forward, all work in the Senior Project course would focus on externally sourced projects.

In the following years, we took on more exploratory projects with a similar structure: build a minimal viable product, experiment with it, figure out what works and what doesn't, and make (often substantial) changes and/or additions. However, we soon realized that there was a new set of problems. The exploratory nature of the projects and deep partner engagement often led to rapidly evolving requirements. Following the agile principle of customer involvement, we require our student teams to have bi-weekly (and sometimes weekly) check-ins with their external project partner to inform them about ongoing work and demonstrate their current functionality. After seeing how their ideas actually work as software, project partners may change their minds about what they want in their product, requiring a quick pivot and substantial reworking of the current product backlog. At the same time, given the many unknowns in such projects (because they are exploratory by nature, or because students may not have a lot of experience with particular technologies), the possibility of delays due to roadblocks is substantial. On the other hand, it was not uncommon for teams to complete all work earlier in the semester and end up with a sprint or two doing nothing due to the same unknowns about the technologies or the complexity of the desired features. All these scenarios lead to the same outcome: it may be difficult to predict the rate of student work, resulting in not being able to complete everything planned or finishing all work too early. The lesson we learned is that it is essential to *prioritize all features in the scope of work* and be reasonably flexible about it. We focus primarily on separating core project goals (expected to be finished in one semester) from stretch goals (OK to leave out), but we also use finer granularity of priorities. For example, sometimes we use three tiers of priorities for some projects: baseline goals (must have, essential to implement this semester), target goals (nice to have, some might be implemented this semester), and stretch goals (aspirational, unlikely to be implemented this semester). Listing these aspirational goals provides student teams with a vision for what this project may become at later.

As we started taking on larger-scale projects, it became evident that we should avoid planning too far into the future. Prioritizing features in longer-term projects is useful, but it is important to avoid being too specific about the longer-term goals of a larger project. A typical project partner would want to see and "play around" with each semester's outcomes before deciding what they want to see next. Following the "inspect and adapt" approach used for continuous evaluation and adjustment in scrum projects, we use the same principle to *re-evaluate and plan the project work for every upcoming semester*. In close collaboration with the project partner, we revise the project scope for each semester in a multi-semester project to reflect completed work and current priorities.

With the first project continuing into the next semester, knowledge transfer between teams became a critical consider-

SPSG practices	Inception	Elaboration	Development	Transition
Knowledge transfer practices				
Strong code documentation		X	X	X
Deployment documentation		X	X	X
Initial support from previous team members	X	X		
Availability of previous team members		X	X	
Student engagement for new teams				
Near-peer mentoring		X	X	X
Structured mentorship		X	X	X
Partner engagement and scope management				
Involving alumni in sourcing projects	X			X
Frequent interaction with partners	X	X	X	X
Partners helping prioritize project features	X	X	X	
Flexible project planning	X	X	X	
Continuous improvement				
Regular sprint retrospectives by students			X	
Stakeholder engagement	X	X	X	X
Post-deployment maintenance practices				
Integrating requests and fixes into next iteration	X			X
Dedicated maintenance project	X			X
Staff engineers maintain environment				X
Project partner handles maintenance				X

"X" indicates this practice is useful/applicable in this phase of the project

Fig. 1: Multi-semester practices applicability timeline

ation. While thorough code and deployment documentation in version control systems GitHub or GitLab was essential, that alone was not sufficient. Therefore, in addition to providing a user manual as a part of their project deliverables, we started requiring each team to also produce a project deployment manual that would provide a detailed walkthrough of configuring the development environment to continue work on the project. Given the limited student experience with system configuration and with writing technical documentation, it became clear after a few ”contactless” team-to-team project handovers that detailed documentation alone is often insufficient. In our experience, the best way to ensure project continuity is through a *balanced combination of well-documented code, detailed deployment and configuration documentation, and personal support*. Typically, we ask one member of the previous team to volunteer about 5 hours to spend with the new team, providing a general overview of the project at a low technical level, as well as an initial walkthrough of the code repository. This time also includes any possible follow-ups with questions related to project configuration, technical architecture, and similar topics.

Due to the underlying curriculum structure, the context of CCSU's Software Engineering Studio also offers additional ways to address project handover between teams. Our curriculum includes two courses, Software Engineering and Senior Project, which form the capstone sequence for our ABET-accredited BS CS program. Both courses include a substantial team-based software engineering project. The Software Engineering course is also a popular program elective among students in our BA CS program. As a result, we initially experimented with starting yearlong projects in the Software Engineering course and completing them in the Senior Project course. However, we abandoned this approach for several reasons. For many of our projects, it is difficult to predict in advance how long they will take—two or three semesters, or even longer. If the project extends beyond two semesters, we would still be facing a handover issue before the third semester. Additionally, in order to *maximize student benefits from these authentic experiences*, we typically aim to switch students from one project to another after the first semester to increase their exposure to different problem domains, different external partners and project stakeholders, and a different set of technical tools (tech stack). Furthermore, we also usually remix all teams after one semester to maximize student exposure to different personality styles and different aspects of team dynamics, and help them experience and resolve a broader range of teamwork-related issues. We believe that this approach has helped students grow into more mature software engineering professionals by helping them build their technical competencies and develop stronger professional dispositions. While we are currently collecting data to validate professional disposition development, preliminary observations suggest positive outcomes.

We believe that there is a benefit in creating teams composed of students from both courses with at least one team member continuing on the same project while enrolled in the second course. This approach would provide a built-in solution for knowledge transfer from one team to the next. However, students continuing on the same project would not be able to benefit from the advantages of working on a different project as described above. The biggest challenge that so far prevented us from experimenting with this approach is the mismatch of project scope, project phases, and sprint cadence between the two courses: due to the requirements of course topic coverage, projects in the two courses start and end at different points in the semester with the Software Engineering project lasting four sprints and the Senior Project having six sprints.

C. Considerations for Multiple Concurrent Teams

Many projects with our project partners start out with a relatively vague idea of solving a particular business or

operational problem. We work with them to understand what the potential solution might be, but very frequently it is not obvious to the partner what type of solution they might prefer. On several projects, we approached this situation by offering a "friendly competition" where two or three student teams work on *different solutions to the same problem* outlined in the partner's project proposal. For example, it could be about using different cross-platform development frameworks for mobile applications, resulting in slightly different sets of application capabilities and/or maintenance requirements. On another recent project, three different teams experimented with three different Large Language Models and their APIs for the most efficient, convenient, and cost-effective way to help format free-form resumes into one or more predefined templates. The teams work in parallel and aim to develop the same amount of higher-priority features. This is supported by regular in-class reporting about their work including weekly check-ins and sprint retrospectives. These regular reports frequently lead to cross-pollination of design ideas among the teams working on the same project. At the semester's end, the project partner has several alternative solutions. Depending on the project's context and the technical needs, one solution may be chosen for further development. Sometimes, all alternatives are treated as rough prototypes and are scrapped, but the knowledge of what works best generated by the teams is used as input to substantially refine and narrow down the project proposal, which is then offered to a new team in the next semester.

We have also experimented with multiple teams working on different functionalities within established projects. The biggest issue we experienced was when the work of one team may be dependent on any outcome of another team. Inevitably, some of the work of one team would be delayed, which would create a significant roadblock for another team. We tried resolving this issue by making sure that the depended-upon features would have high priority and would be placed in earlier sprints. We also tried making sure that the other team would have enough flexibility to work on other features assuming that there would be delays. The only workable solution was to *minimize (and ideally eliminate) dependencies by scoping each team's work so they are completely independent*. This approach has been successful in several projects up to three teams working in parallel, each making quality contributions.

D. Project Maintenance Considerations

Post-deployment maintenance is a crucial concern that must be discussed with the project partner from the outset. We are always upfront with our project partners that unless there is a team actively working on that project in a given semester, we will not always be able to offer quick bug fixes. Additional considerations that need to be addressed at the same time include any expenses related to project hosting, as well as license fees, monitoring the environment, the need for potential tech support if outside vendors make changes, potential security issues, etc. There are a number of approaches we offer to our partners that may be more or less suitable depending on the partner's organizational and project context.

The most common approach to the maintenance of a project that is actively being used by the partner is to accumulate a list of all bug fixes observed over several months of use. Often, after experimenting with the produced solutions, partners realize they may want certain features implemented differently and/or new features added. In such a scenario, which is very typical for many of our projects, the list of bug fixes can be rolled into a project iteration together with requests for changes and/or new functionality. This typically results in a semester gap in active work being done on the project, but this is usually not a concern for many of our partners.

Some of our partners do have technical personnel on staff who may be capable and willing to address bug fixes. For partners that are lacking such staff and for projects that cannot afford to wait several months for bug fixes, we always advise them to work with one or two members of the student team(s) who would be willing to address any emerging software faults as they come up. The expectation is that such work can be done pro bono or for some compensation provided by the partner when the student is no longer a part of that team once the corresponding course is over.

This narrative reflects the evolution of practices over more than a decade at The Studio, forming the basis forthe SPSG framework. However, it is just one story. Each author has faced a variety of challenges unique to their institutional contexts, we have also observed many similarities in these challenges and the practices that have proven effective across our institutions. The result of the combined experience has led to a set of best practices for handling challenges in a variety of contexts.

IV. Project Transition Practices

In the context of multi-semester service learning projects, certain practices have emerged as reliable solutions that are effective in addressing common challenges. These practices, similar to design patterns in software engineering, have been validated through repeated application across multiple projects, partners, and institutional settings. By identifying and adopting these proven practices, educators can tailor them to their specific institutional contexts, thereby enhancing the continuity, scalability, and overall impact of their service-learning initiatives. This section outlines a collection of best practices refined through extensive experience, offering a flexible framework for educators to implement and adapt in their unique environments.

A. Knowledge transfer practices

Effective knowledge transfer and handoff is crucial for the success of multi-semester projects. Robust documentation protocols and organized code repositories ensure that incoming teams can quickly understand the project's current state and continue development seamlessly.

Challenges encountered: Managing the handoff process between teams presented several challenges. New teams often struggled to understand the existing codebase and project context, leading to delays and inefficiencies. Without robust documentation, incoming teams faced difficulties in setting

up their development environments and navigating technical complexities. Additionally, the lack of personal support from previous team members hindered the transfer of critical knowledge, resulting in a steep learning curve for new teams. The availability of previous team members in a multi-course sequence also posed challenges, as it limited students' exposure to different projects and technical challenges.

Solutions identified:

***Strong code documentation**:* Comprehensive code documentation became fundamental for new teams to understand the project's current state and continue development seamlessly. This includes detailed comments within the code, clear explanations of the project's architecture, and guidelines for navigating the codebase. By ensuring thorough documentation, new teams could quickly get up to speed and continue development without significant setbacks.

***Deployment documentation**:* Deployment guides were created to provide step-by-step instructions for configuring the necessary tools and dependencies, ensuring that new teams could quickly get started without facing technical hurdles. Detailed deployment guides helped new teams avoid initial technical challenges and focus on development tasks.

***Initial support from previous team members**:* Personal support from members of the previous team significantly enhanced the handoff process. Typically, one member of the previous team volunteered to spend a few hours with the new team, offering a general overview of the project and an initial walkthrough of the code repository. This support included answering questions related to project configuration, technical architecture, and other relevant topics. By providing limited initial support, new teams could overcome challenges more effectively and continue development with greater confidence.

***Availability of previous team members**:* In some cases, the same team or part of the same team continued work in the next semester, providing continuity and reducing the learning curve for new members. This approach ensured that the project benefited from the accumulated knowledge and experience of the previous team. However, it was essential to balance this with opportunities for students to work on different projects and develop a broader range of skills.

***New team with background information**:* Alternatively, a new team picked up the project in the second course, but they already had the background information from the previous semester. This method allowed fresh perspectives and ideas to be brought into the project while leveraging the foundational knowledge provided by the previous team. This approach led to innovative solutions and improvements in the project, as new teams brought fresh perspectives and ideas.

B. Post-deployment maintenance practices

Post-deployment maintenance is a particularly crucial aspect of multi-semester service-learning projects, ensuring that the software remains functional and continues to meet the needs of the project partner. Effective maintenance practices are essential to address any issues that arise after the initial deployment and to implement any necessary updates or improvements.

Challenges encountered: Managing post-deployment maintenance presented several challenges. One of the primary difficulties was ensuring that there was a dedicated team or individual responsible for handling maintenance requests. Without a clear plan for maintenance, issues such as defects and feature requests could accumulate, leading to a decline in the software's usability and effectiveness. Additionally, coordinating maintenance efforts with the project partner and ensuring that they had the necessary resources and support to address any issues was challenging. The availability of past team members to handle maintenance, either on a contract or pro bono basis, also posed logistical and financial challenges.

Solutions identified:

***Dedicated maintenance team**:* Establishing an existing team reserved for handling maintenance requests ensures that there is always a group of individuals responsible for addressing any issues that arise post-deployment. This team can be composed of students who have previously worked on the project or new students who are trained to handle maintenance tasks.

***Integrating requests and fixes into next iteration**:* Accumulating maintenance requests and bug fixes and incorporating them into a subsequent project iteration provides a structured approach to maintenance. This method ensures maintenance tasks are addressed systematically and new features or improvements are implemented in a cohesive manner.

***Project partner handling maintenance**:* In some cases, the project partner may have one of the past team members handle maintenance on a contract or pro bono basis. This approach leverages the existing knowledge and experience of past team members, ensuring that maintenance tasks are handled efficiently and effectively.

***Staff engineers maintaining the environment**:* Having an agreement for university staff engineers to maintain the product environment for tasks such as monitoring and security updates can provide a reliable and consistent approach, making it easier for a deployed system to remain stable until the next semester when it is taken up again. Staff engineers at the partner organization can support post-deployment maintenance by ensuring that the software remains functional and up-to-date, addressing any issues that arise, and implementing necessary updates or improvements.

C. Partner engagement and scope management practices

Effective partner engagement and scope management are crucial for the success of multi-semester service-learning projects. Engaging with external partners helps align goals and expectations, ensuring that both parties understand academic constraints and project requirements. Managing the scope of these projects involves prioritizing features and maintaining flexibility about longer-term goals.

Challenges encountered: Engaging with external partners posed challenges. Partners often had limited experience with the software development process, misaligning goals and expectations. Frequent interactions were necessary to ensure that both parties understood academic constraints, especially concerning project cadence and timeframes. Additionally, manag-

ing the scope of multi-semester projects was challenging due to the rapid evolution of project requirements. Prioritizing features and adapting to new requirements. Regular collaboration with project partners to revise scope is essential for aligning with current priorities and completed work.

Solutions identified:

Frequent interactions with partners: Regular communication with external partners helps align goals and expectations, ensuring that both parties understand academic constraints and project requirements. This practice minimizes potential challenges and fosters a collaborative environment.

Helping prioritize project features: Separating core project goals from stretch goals allows teams to focus on essential tasks while providing a vision for future development. Using finer granularity of priorities, such as baseline goals, target goals, and stretch goals, helps manage the scope effectively.

Flexible project planning: Avoiding overly specific long-term goals and using the "inspect and adapt" approach for continuous evaluation and adjustment ensures that projects remain aligned with current priorities and completed work. Regular collaboration with project partners to revise the project scope enhances scalability and sustainability.

Involving alumni in sourcing projects: Engaging alumni in sourcing projects provides the partner with valuable insights and helps identify suitable projects that align with academic constraints and student capabilities. Alumni involvement also fosters a sense of community and continuity.

D. Practices for student engagement for new teams

Student engagement practices focus on motivating and retaining students throughout the project's lifecycle. Integrating service-learning projects into the curriculum provides valuable experiential learning opportunities, enhancing student motivation and learning outcomes. Structured near-peer mentoring helps students build confidence and develop non-technical skills, such as effective problem-solving and teamwork.

When students first begin their journey in software engineering, we observed that they often encounter a range of challenges and struggles. These difficulties are a normal part of the learning process and include understanding complex concepts, navigating uncertainty, lacking confidence in their abilities, and developing effective problem-solving skills.

Challenges encountered: Many students find it challenging to grasp complex software engineering concepts and methodologies, leading to frustration and a lack of confidence in their abilities. Building confidence is another significant challenge. The initial stages of learning can be daunting, and students may doubt their ability to succeed. Several students even expressed feelings of impostor syndrome. Furthermore, students often need guidance on when some struggle is appropriate as they learn on their own versus when to ask for help and how to elevate dependencies or roadblocks in a timely manner.

Solutions identified:

Near-peer mentoring: Interacting with peers or mentors who have successfully navigated similar challenges can be incredibly reassuring. Over the years, various models for incorporating this practice have been successfully tried, such as recruiting recent alumni or graduate students who have been through similar project experiences [9].

Structured mentorship: One approach has been having near-peers serve as scrum masters without providing technical assistance. This way, students receive mentorship that helps them build their confidence and skills to be successful, and they are better able to see that person as a mentor and resource in these other areas, rather than just thinking of them as someone to go to with technical problems.

E. Continuous improvement practices

Continuous improvement is a vital aspect of multi-semester projects, aimed at enhancing student outcomes and ensuring project partners perceive the projects as successful. Implementing feedback loops and data collection practices allows educators to monitor progress, identify areas for enhancement, and make informed decisions to improve the overall effectiveness of the projects.

Challenges encountered: Due to the unique nature of every project and team dynamic, if regular check-ins are not part of the process, it is very easy for teams to struggle or hit roadblocks that can jeopardize the achievement of course learning outcomes. Additionally, we found that students' understanding of priorities often differed from those of project partners, particularly with less experienced student teams and service-learning partners who had little experience with the software development process.

Solutions identified:

Regular sprint retrospectives by students: Conducting regular sprint retrospectives allows student teams to reflect on their progress, identify areas for improvement, and celebrate successes. These retrospectives also provide instructors with valuable insights into team dynamics, project challenges, and individual contributions.

Stakeholder engagement: Engaging project partners in the feedback process ensures that their perspectives are considered, helping to align project goals with their expectations and needs. In our projects, we have found regular student check-ins, either weekly or every other week, to work well for this. Additionally, instructors also check in periodically to get feedback on the project's progress from the partners' perspective to understand changes that may be needed.

V. Evaluation

Evaluation of multi-semester service-learning strategies employed a qualitative, non-participant, naturalistic observational approach guided by "Big-Tent" criteria [32]. An external observer, Williams, systematically collected detailed notes across three projects over two academic semesters, focusing on key interactions: weekly team planning, external partner engagements, and in-class sprint retrospectives. This enabled real-time capture of intricate team dynamics, communication, iterative problem-solving, and nuanced stakeholder interactions. The non-participating choice minimized researcher influence and ensured authentic data [33], fostering objectivity and reducing bias. This qualitative approach was foundational

given the phenomena's inherent complexity and human-centric nature, uniquely suited for rich, contextual, non-quantifiable data to understand *how* and *why* events occur. It ensured high ecological validity by capturing authentic behaviors in natural settings, paramount for dynamic, real-world project-based learning [34]. Acknowledging qualitative inquiry's inherent subjectivity, observer Williams (a Caucasian male with prior industry and academic software engineering experience) engaged in regular memoing and reflexive note-taking. Though not their instructor, his departmental faculty role might have subtly shaped student engagement. This reflexivity, crucial for rigor, surfaced assumptions and maintained attentiveness to how personal background—including experience with process rigor, communication, and commitment to gender diversity (one female-identifying student observed per team)—could influence data interpretation. This systematic reflexivity strengthens findings' validity by transparently addressing researcher positionality and potential analytical impact (Table I).

It covered three projects across two semesters. The observer attended three types of meetings: weekly team planning meetings, weekly meetings with the project partner, and sprint retrospectives presented in class. Additionally, the observer checked in with the teams throughout the semester regarding challenges and successes. Findings are organized into three perspectives: team communications, project partner interactions, and near-peer mentor meetings.

Team communication*:* Throughout the semester, teams engaged in regular meetings to discuss their progress, challenges, and solutions. They indicated that thorough code documentation and deployment instructions from the previous semester's project were incredibly helpful in setting up a working local development environment and understanding the code. Despite this, the new team initially found the process slow. However, a student from the previous team met with the new team to provide insights into the project's vision and guidance on navigating the codebase, which facilitated a smoother transition and enhanced the team's understanding of the project.

Project partner interactions*:* The interactions between the teams and the project partner were crucial in shaping the project's direction. The partner's feedback during meetings played a significant role in prioritizing tasks and addressing challenges. For instance, the second-semester team was tasked with fixing bugs from the previous team's app while implementing new functionality. The partner's decision to prioritize impactful new features over less critical bug fixes provided the team with a real-world understanding of balancing stakeholder needs and time constraints. This was eye-opening for the team, as they realized the importance of aligning project goals with the partner's priorities. One student remarked that they had never considered these types of decisions being made for any reason other than profit. Another example of the value of frequent partner interactions was when the team presented the partner with a choice of two technical solutions to solve a requirement. The partner pushed back, stating that the choice was meaningless without more context. This required the team to rethink and re-present the choice, articulating it in terms of the desired end-user experience, greatly improving their ability to convey technical concepts in a more accessible manner.

Near-peer mentor meetings*:* Across the three teams, a near-peer mentor's involvement, particularly during the initial weeks, was instrumental in building the team's confidence and guiding them through the project's challenges. The mentor shared insights from their previous experience, highlighting effective communication practices within the team and with the project partner. Although the mentor's interaction was limited due to external commitments, the initial meetings were sufficient to set the team on the right path. The mentor's advice on articulating items that needed input from the project partner helped the team prioritize their work effectively and fostered a collaborative environment. Additionally, the presence of a previous team member during the first two weeks was a huge accelerant in speeding up knowledge transfer. This individual provided context and answered questions, which significantly enhanced the new team's ability to quickly understand and build upon the previous semester's work.

VI. Discussion and future work

Our experiences with multi-semester service-learning projects have highlighted several key takeaways. By addressing common pitfalls and leveraging transferability evidence, we believe a framework incorporating these practices can significantly enhance the educational impact of such projects.

The strategies outlined in this paper have been successfully tried across a range of institutions, providing strong evidence of their transferability. The use of the SPSG framework, which integrates these practices into a larger service-learning framework, has yielded positive results at Central Connecticut State University and Xavier University. This success demonstrates the potential for these practices to be adapted and applied in various academic contexts. Future work will study the application of these practices in additional institutions to refine and identify best practices for unique institutional challenges not encountered by the authors. To further assess the effectiveness of these practices, we are gathering feedback from students, mentors, and project partners to evaluate their impact on project outcomes and learning experiences. The insights gained from this assessment will inform future iterations of the SPSG framework and help us identify areas for improvement.

References

[1] G. Tembrevilla, A. Phillion, and M. Zeadin, "Experiential learning in engineering education: A systematic literature review," *Journal of Engineering Education*, vol. 113, no. 1, pp. 195–218, 2024.

[2] W. Haque, "Experiential learning: Beyond the classroom and connecting with the industry," in *Tomorrow's Learning: Involving Everyone. Learning with and about Technologies and Computing*, A. Tatnall and M. Webb, Eds. Springer Int. Publishing, 2017, pp. 443–452.

[3] A. Bloomfield, M. Sherriff, and K. Williams, "A service learning practicum capstone," in *Proceedings of the 45th ACM Technical Symposium on Computer Science Education*. Atlanta Georgia USA: ACM, Mar. 2014, pp. 265–270.

[4] F. Robledo Yamamoto, L. Barker, and A. Voida, "CISing Up Service Learning: A Systematic Review of Service Learning Experiences in Computer and Information Science," *ACM Transactions on Computing Education*, vol. 23, no. 3, pp. 1–56, Sep. 2023.

Observational Focus Area	Key Behaviors/ Interactions Observed	Corresponding Best Practice/ Strategy Validated	Illustrative Qualitative Insight/ Finding
Team Planning Meetings	Internal coordination, Problem-solving, Task division, Conflict resolution	Robust Documentation Protocols, Organized Code Repositories	"Thorough code documentation and deployment instructions...incredibly helpful."
Project Partner Interactions	Client communication, Requirements negotiation, Scope alignment, Feedback integration	Frequent Interactions with Partners, Helping Prioritize Project Features	"Partner's decision to prioritize impactful new features...real-world understanding."
Sprint Retrospectives	Team reflection, Continuous improvement, Learning from failures	Continuous Feedback Loops, Flexible Project Planning	"Initially amount of change was overwhelming...prioritized with client for sprint...practice flexibility"
Near-Peer Mentor Sessions	Knowledge transfer, Confidence building, Professional skill development, Guidance on roadblocks	Mentorship Systems, Structured Mentorship	"Near-peer mentor's involvement... instrumental in building the team's confidence."
Team Check-Ins with Observer	Identification of challenges/successes,	Initial support from previous team members	"In 15 minutes, we solved what I had been struggling with for hours."

TABLE I: Observed Data Points and Their Significance in Evaluation

[5] K. Timmerman and M. Goldweber, "Department-wide multi-semester community engaged learning initiative to overcome common barriers to service-learning implementation," in *Proc. 53rd ACM Tech. Symp. Comput. Sci. Educ. - Volume 1*, ser. SIGCSE 2022. New York, NY, USA: ACM, 2022, p. 808–814.

[6] C. N. Bull and J. Whittle, "Supporting reflective practice in software engineering education through a studio-based approach," *IEEE Software*, vol. 31, no. 4, pp. 44–50, 2014.

[7] R. Raj, M. Sabin, J. Impagliazzo, D. Bowers, M. Daniels, F. Hermans, N. Kiesler, A. N. Kumar, B. MacKellar, R. McCauley, S. W. Nabi, and M. Oudshoorn, "Professional competencies in computing education: Pedagogies and assessment," in *Proc. 2021 Work. Group Rep. on Innovation and Technol. Comput. Sci. Education*, ser. ITiCSE-WGR '21. ACM, 2022, p. 133–161.

[8] V. Garousi, G. Giray, E. Tuzun, C. Catal, and M. Felderer, "Closing the gap between software engineering education and industrial needs," *IEEE Software*, vol. 37, no. 2, pp. 68–77, 2020.

[9] J. Davis and S. A. Rebelsky, "Developing soft and technical skills through multi-semester, remotely mentored, community-service projects," in *Proc. 50th ACM Tech. Symp. Comput. Sci. Educ.*, ser. SIGCSE '19. ACM, 2019, p. 29–35.

[10] S. Kurkovsky, "Student reflections on service-learning in software engineering and their experiences with non-technical clients," in *Proc. ACM Conf. Global Comput. Educ.*, ser. CompEd 2023, vol. 1. New York, NY, USA: ACM, 2023, p. 112–118.

[11] N. Herbert, "Reflections on 17 years of ICT capstone project coordination: Effective strategies for managing clients, teams and assessment," in *Proc. 49th ACM Tech. Symp. Comput. Sci. Educ.*, ser. SIGCSE '18. New York, NY, USA: ACM, 2018, p. 215–220.

[12] P. Flener, "Realism in project-based software engineering courses: Rewards, risks, and recommendations," in *Comput. Inf. Sci. – ISCIS 2006*, A. Levi, E. Savaş, H. Yenigün, S. Balcısoy, and Y. Saygın, Eds. Springer Berlin Heidelberg, 2006, pp. 1031–1039.

[13] B. Krogstie and B. Bygstad, "Cross-community collaboration and learning in customer-driven software engineering student projects," in *20th Conf. on Softw. Eng. Educ. & Training (CSEET'07)*, 2007, pp. 336–343.

[14] G. Braught, J. Maccormick, J. Bowring, Q. Burke, B. Cutler, D. Goldschmidt, M. Krishnamoorthy, W. Turner, S. Huss-Lederman, B. Mackellar, and A. Tucker, "A multi-institutional perspective on H/FOSS projects in the computing curriculum," *ACM Trans. Comput. Educ.*, vol. 18, no. 2, Jul. 2018.

[15] S. Kurkovsky, "Using scaffolding to simplify foss adoption," in *Proc. 27th ACM Conf. Innovation Technol. in Comput. Sci. Educ.*, ser. ITiCSE '22, vol. 2. New York, NY, USA: ACM, 2022, p. 587–588.

[16] S. Seifer and K. Connors, *Faculty Toolkit for Service-Learning in Higher Education*. Learn and Serve America's National Service-Learning Clearinghouse, 2007.

[17] M. Papadimitriou, "High school students' perceptions of their internship experiences and the related impact on career choices and changes," *Online J. for Workforce Educ. Develop.*, vol. 7, no. 1, p. 8, 2014.

[18] N. Abu-Mulaweh and W. Oakes, "Student learning in computing-based service-learning," in *2018 IEEE Frontiers in Educ. Conf. (FIE)*. IEEE, 2018, pp. 1–7.

[19] P. K. Linos, S. Herman, and J. Lally, "A service-learning program for computer science and software engineering," in *Proc. 8th Annu. Comput. Sci. on Innovation and Technol. in Comput. Sci. Educ.*, 2003, pp. 30–34.

[20] F. Dan Richard, B. Berkey, and H. M. Burk, "Motivation and orientation: Faculty perspectives on development and persistence in service learning and community engagement." *J. of Community Engagement & Higher Educ.*, vol. 14, no. 1, 2022.

[21] B. Pérez and Á. L. Rubio, "A project-based learning approach for enhancing learning skills and motivation in software engineering," in *Proc. 51st ACM Tech. Symp. Comput. Sci. Educ.* ACM, Feb. 2020, pp. 309–315.

[22] J. Payton, T. Barnes, K. Buch, A. Rorrer, and H. Zuo, "The effects of integrating service learning into computer science: An inter-institutional longitudinal study," *Comput. Sci. Educ.*, vol. 25, no. 3, pp. 311–324, 2015.

[23] M. Kilkenny, C. L. Hovey, F. Robledo Yamamoto, A. Voida, and L. Barker, "Why should computer and information science programs require service learning?" in *Proc. 53rd ACM Tech. Symp. Comput. Sci. Educ. V. 1*, 2022, pp. 822–828.

[24] M. Salam, D. N. Awang Iskandar, D. H. A. Ibrahim, and M. S. Farooq, "Service learning in higher education: A systematic literature review," *Asia Pacific Education Review*, vol. 20, no. 4, pp. 573–593, 2019.

[25] D. Bargar, A. Gordon, J. Plumblee, J. Ogle, C. Dancz, and D. Vaughn, "Increasing Student Development Through Multi-Level Immersive Learning: Clemson Engineers for Developing Countries Case Study," *Int. J. Service Learn. in Eng., Humanitarian Eng. and Social Entrepreneurship*, vol. 11, no. 2, pp. 55–71, Oct. 2016.

[26] T. T. Yuen, "Scrumming with Educators: Cross-Departmental Collaboration for a Summer Software Engineering Capstone," in *2015 Int. Conf. Learning Teaching in Comput. Eng.* IEEE, Apr. 2015, pp. 124–127.

[27] M. L. Fioravanti, B. Sena, L. N. Paschoal, L. R. Silva, A. P. Allian, E. Y. Nakagawa, S. R. Souza, S. Isotani, and E. F. Barbosa, "Integrating project based learning and project management for software engineering teaching: An experience report," in *Proc. 49th ACM Tech. Symp. on Comput. Sci. Educ.* ACM, Feb. 2018, pp. 806–811.

[28] K. Börner, O. Scrivner, M. Gallant, S. Ma, X. Liu, K. Chewning, L. Wu, and J. A. Evans, "Skill discrepancies between research, education, and jobs reveal the critical need to supply soft skills for the data economy," *Proc. Nat. Acad. Sci.*, vol. 115, no. 50, pp. 12 630–12 637, Dec. 2018.

[29] S. Kurkovsky, "Managing scope in service learning projects," in *Proc. 27th ACM Conf. on Innovation and Technol. in Comput. Sci. Educ. Vol. 2*, ser. ITiCSE '22. New York, NY, USA: ACM, 2022, p. 620.

[30] S. Kurkovsky, C. A. Williams, M. Goldweber, and N. Sommer, "External projects and partners: Addressing challenges and minimizing risks from the outset," in *Proc. 2024 Innovation and Technology in Computer Science Education V. 1 (ITiCSE 2024), July 8–10, 2024, Milan, Italy*, ser. ITiCSE '24. New York, NY, USA: ACM, 2024.

[31] SPSG Hub. (2025) Scaffolded projects for the social good. [Online]. Available: https://spsg-hub.github.io/

[32] S. J. Tracy, "Qualitative quality: Eight "big-tent" criteria for excellent qualitative research," *Qualitative inquiry*, vol. 16, no. 10, pp. 837–851, 2010.

[33] P. Lenberg, R. Feldt, L. Gren, L. G. Wallgren Tengberg, I. Tidefors, and D. Graziotin, "Qualitative software engineering research: Reflections and guidelines," *Journal of Software: Evolution and Process*, vol. 36, no. 6, p. e2607, Jun. 2024.

[34] C. Seaman, "Qualitative methods in empirical studies of software engineering," *IEEE Transactions on Software Engineering*, vol. 25, no. 4, pp. 557–572, Jul. 1999.