Teaching Software Engineering with Project-Based Learning: A Four Years Experience Report

Lina Garcés
Software Engineering Laboratory, Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo - LabES/ICMC/USP
São Carlos, SP, Brazil
linagarces@usp.br

Brauner Oliveira
Software Engineering Laboratory, Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo - LabES/ICMC/USP
São Carlos, SP, Brazil
brauner@usp.br

ABSTRACT

Project-Based Learning (PjBL) is an educational design methodology embraced by instructors within engineering programs and, recently, in software engineering courses. PjBL enables students to confront real-world scenarios and apply their knowledge to create practical and meaningful software products, all while enhancing essential soft skills required in the software industry. This study draws upon the authors' firsthand experiences implementing PjBL across four software engineering courses. It outlines the challenges encountered during each course iteration, emphasizing areas ripe for improvement in future offerings. Insights gathered from four years of teaching with PjBL are shared, detailing the positive and negative outcomes of employing various strategies to address challenges identified in prior course runs. The experience reported herein gives fellow software engineering educators valuable insights, enabling them to avoid ineffective approaches and integrate successful ones into their teaching endeavors.

CCS CONCEPTS

• **Software and its engineering** → *Agile software development*; **Software development techniques**; • **Social and professional topics** → **Computing education**.

KEYWORDS

Software Engineering, Problem-Based Learning, PjBL, Agile

1 INTRODUCTION

Teaching and learning software engineering nowadays remains challenging due to the relevant and pervasive role of software in our society. The way we develop software has changed several times over the last decades with the introduction of different ideas, methods, paradigms, and technologies. Beyond the large set of concepts required to understand software engineering, the development, and improvement of students' soft skills is a highly desired outcome of educational experiences [4, 12]. Consequently, software engineering programs and courses must continuously adapt their content and learning strategies to keep on par with software industry dynamics.

As a way to overcome these challenges, Project-Based Learning (PjBL) has been largely employed for software engineering education [1, 14, 15, 17]. PjBL is considered a student-centered educational method in which a project is developed as a main part of a course [15]. Software engineering topics addressed in lectures and classroom activities support groups of students while developing a software project that aims to solve real or realistic problems [15], which may involve actual stakeholders. However, PjBL can be configured and employed in different ways [1, 17], affecting the experience of students [17] and possibly its learning outcomes. Studies have demonstrated that PjBL contributes to long-term retention, soft-skills development, and satisfaction of students and teachers when compared with traditional approaches [16].

While several studies recognize the benefits of PjBL, implementing it imposes different challenges on the instructor, who has to encourage and keep students engaged throughout the project, provide feedback continuously, and evaluate students [7]. In this sense, this paper describes a 4-year experience employing PjBL to teach software engineering to students enrolled in two courses of the Computer Engineering and Information System undergraduate programs. The way PjBL was applied changed over the years to address issues identified during each course run. Pre- and post-course questionnaires were applied as part of the course design, the first to support our decisions on topics to be addressed and the latter to assess the students' perspective on several aspects after the course. The contribution of this paper lies in sharing the PjBL-oriented course design, the rationale behind it, and the lessons learned from applying PjBL. Moreover, this study's findings are contrasted with related experiences of conducting PjBL for software engineering education.

The remainder of this paper is organized as follows. **Section 2** presents an overview of related work and positions contributions of this experience report. **Section 3** provides the course design offered over the four years. **Section 4** describes each course run and their relevant details such as their design and issues faced. The results of students' responses to questionnaires are discussed in **Section 5**, while the lessons learned are described in **Section 6**. Finally, **Section 7** outlines the conclusions and future work.

2 RELATED WORK

Currently, there are various experiences related to the utilization of PjBL in software engineering courses. Most reports focus on lessons from implementing PjBL in introductory [15] and advanced [2, 13, 17, 18] software engineering courses. Additionally, there are works discussing experiences gained from PjBL in courses spanning project management [5, 8], continuous delivery [9], object-oriented analysis and design [12], user experience (UX) [3], and recently, open-source software engineering [6]. Despite most of the studies highlight experiences in undergraduate level, notable instances of

PjBL in graduate courses [5, 6, 17] and industry training [8, 9] are also documented.

Typically, PjBL-oriented courses have a duration of a full semester, lasting between 12 to 16 weeks or totaling 30 to 60 hours. There are exceptions, with two studies reporting PjBL implementation within shorter timeframes, such as 10 weeks (40 hours) [2], 8 weeks (16 hours) [12], or even just 3 days (20 hours) for industry training [8].

The majority of studies report experiences within face-to-face courses, with the exception of Flach and Feitosa [6], Suciu et al. [18], which present results on applying PjBL in fully online courses. Regarding theory contents, all studies divide course time between lessons and project development. Only two courses [9, 18] destined full-time for project execution. All studies combined PjBL with at least one additional activity, including workshops, flipped classrooms, lab activities, exams (e.g., individual tests or quizzes), recommended readings, invited talks, classroom discussions, and traditional lectures.

Some studies have involved industry partners or practitioners [2, 8, 9, 13, 17, 18]. Practitioners' roles within team projects vary, ranging from proposing project ideas to mentoring and acting as a client or product owner. Few studies [8, 9, 13, 17] actively engaged real end users or customers as part of the team. In other studies [5, 6], the customer role was performed by senior undergraduate or postgraduate students.

The experience time reported in the related studies varies between one and seven years of PjBL practice. An exception is the study of Ståhl et al. [17], which presents a consolidated experience dating back to 1977.

From this perspective, the experience reported in this work shares a similar course design and application of PjBL to related work. However, notable differences include the diversity of course settings, e.g., course levels (introductory and advanced), implementation across two different universities, different project durations (8 and 16 weeks), formats (online and face-to-face), team autonomy levels, and the presence or absence of real customers. In all cases, ongoing feedback from students and customers served as the foundation for subsequent course improvements. Additionally, we discuss findings that corroborate some challenges and lessons learned as identified in related studies.

3 COURSE DESIGN

The course design has evolved over the years as a consequence of the lessons learned with each run and the restrictions imposed by ERT (Emergency Remote Teaching) due to Covid-19 outbreak. Additional factors, such as students' background in software development and knowledge of software engineering topics, also impacted the design. The four-year experience described here took place in the context of two different courses, one offered as part of the Computer Engineering Program of University A and the other offered by University B for students of the Information Systems program. Both courses are mandatory for such undergraduate programs.

The first course, *Software Engineering*, is offered annually to 7th-semester students by University A and consists of 120 hours of classes and practice. This course has the following hard requirements: Programming fundamentals, object-oriented programming, and object-oriented analysis and design. Web development and Database II are optional requirements. The second course, *Software Development in Information Systems*, is also offered annually to 7th-semester students by University B and consists of 128 hours of classes and practice. In order to enroll, students have to complete the courses Programming fundamentals, Software Engineering I, Object-Oriented Programming, Object-Oriented Analysis and Design, Web development, and Database I. Optional requirements are Software Engineering II and Database II. In the first course, students are fully dedicated, whereas, in the second course, part of the students split their time between full-time jobs and their undergraduate studies.

Since we employed PjBL, class students formed their own teams to develop one software project. SCRUM was selected as a model for project development. Therefore, each course run consisted of a series of sprints and deliverables that would culminate in a minimum viable software product (MVP) at the end of the project. The number of sprints, the deliverables, and the schedule are determined at the beginning of the project and are the same for all teams. Students select at least one software engineering role (e.g., analyst, developer, tester, architect, and manager), mimicking a real software development team. Project meetings with the professor are also a core activity of this course design, which intends to continuously assess project progress and provide feedback whenever required. Groups are evaluated regarding the accomplishment of deliveries, considering completeness, quality, estimated accomplishment, deadlines, and customers' satisfaction with the product's final release. During sprint review meetings, teams must present delivery results to the whole class and, in some sprints, to end users.

To support the groups, different teaching methods, such as flipped classroom, laboratory activities, case-based learning, and industry expert-invited talks, were employed in addition to lectures on several topics related to software engineering. Moreover, individual tests are defined to assess each student's knowledge of core software engineering topics and their practical application. The **Figure 1** provides a detailed overview of the main components of each course run. The topics covered as part of theory are listed and described in **Table 1**.

During course runs, learning management systems (LMS) such as Moodle and SIGAA were used to organize academic material (e.g., slides, video courses, reading material, links, tasks, etc.), receive project deliverables submissions, facilitate communication with students, and record grades. Moreover, technologies such as Slack or Discord were adopted to provide students with a collaborative environment for project development and effective communication channels with the teacher. Finally, tools such as Notion and spreadsheets were used to register and share the teacher's qualitative assessment of teams or feedback to teams.

4 COURSE RUNS

This section describes our experience with each course run. The number of students enrolled and their prior experience in certain areas is shown in **Table 2**. Overall, a considerable amount of students had some experience in the software industry. The most popular

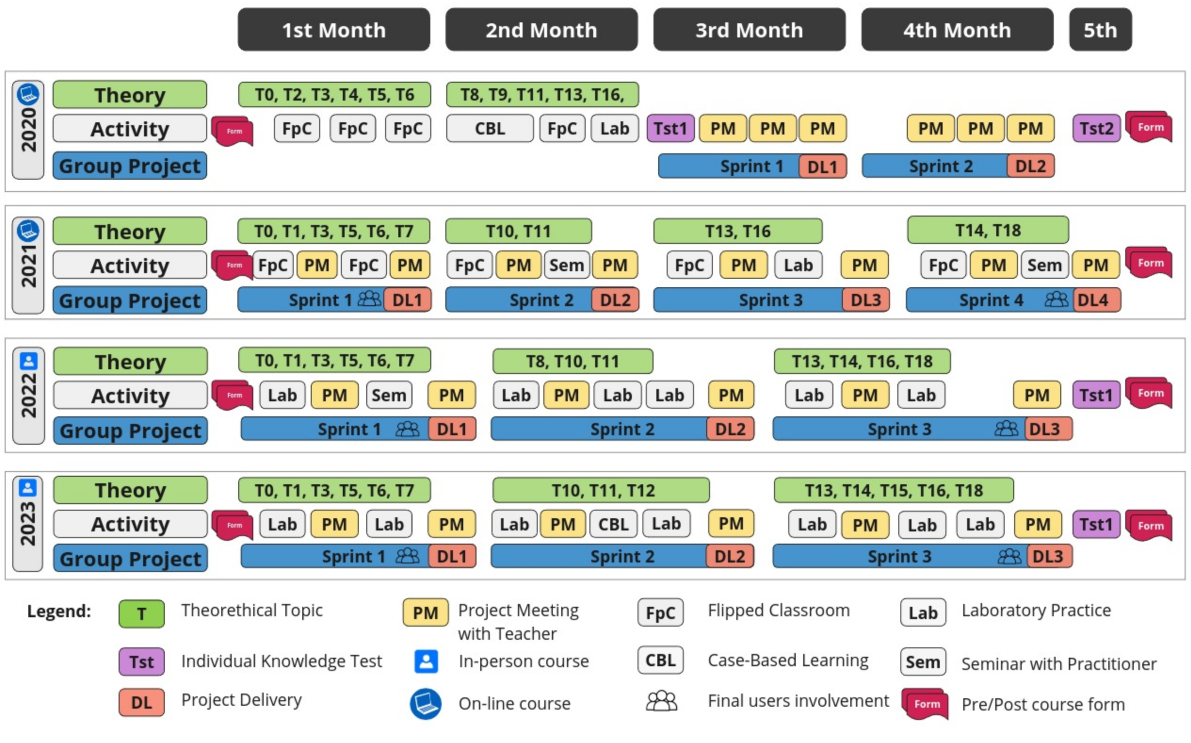

Figure 1: Scheme of software engineering course evolution over four years

Table 1: Software Engineering Topics

ID	Topic	Description
T0	Course's Introduction	Intends to know the professor and students and present the course design. Groups are formed
T1	Design Thinking	Presents the purpose, utility and iterative process of design thinking. A real application case of design thinking is presented to the students. Students' groups are encouraged to execute the design thinking process to define the software product they will develop as a course project
T2	Software development process models	Exposes the prescriptive models of software development processes, such as waterfall, V, spiral, evolutive prototyping, and rational unified process models
T3	Agile software development process models	Introduces the agile manifesto and agile models as SCRUM, XP and Crystal. A special focus is given to SCRUM, the pre-defined model to be used by students teams in the course project.
T4	Software process improvement	Presents the importance of continuous improvement in software development, studying the general SPI model and the CMMI and MPS-BR models.
T5	Agile project management	Focuses on concepts of project management, presenting planning and estimation techniques as SLOC/KSLOC (Lines of Code), FPA (Function Points Analysis), and USP (User Story Points). In particular, the Planning Poker technique and team velocity metrics are studied and applied in-class exercises. Teams are requested to use USP, planning poker, and velocity metrics in their projects.
T6	Requirement Engineering	Studies requirement concepts types, and requirements elicitation, specification, and validation techniques. Specifically, teams are asked to use User Stories (US), behavioral scenarios, and quality attributes scenarios to specify functional and non-functional requirements for their software products.
T7	Interviews	Presents the good practices for planning, executing and reporting interviews with final users. Also, techniques for extracting requirements from interviews are taught. Teams must conduct interviews with real final users as elicitation technique in their projects.
T8	Product quality	Familiarises the students with quality attributes concepts. The ISO/IEC 25010 quality model is presented as the definitions for each quality characteristic and sub-characteristic of this model. Teams must identify the most relevant quality attributes requirements for their products, e.g., security, availability, maintainability, integration, etc.
T9	Software architecture	Introduces the main concepts of software architecture and the stages of the architectural process, i.e., analysis, design, and evaluation. The concepts of reference architectures, architectural styles and patterns, and architectural tactics are also taugth.
T10	Software architecture for the web	Brings a panorama of software architectures evolution for web systems, e.g., client-server, n-tier, SOA, microsservices, and clean and hexagonal architectures, and their benefits and limitations
T11	Architecture representation	Introduces the concepts of software architectures documentation, views, viewpoints, and diagraming. Specifically, it focus on the C4 model presenting some application examples.
T12	Architectural Decision Records	Describes techniques to document architectural decisions making use of ADRs templates and tools
T13	Software testing	Introduces the main concepts of software testing, such as, testing levels (i.e., unit, integration, and system), testing techniques (e.g., white-box, black-box, dynamic, and static), purpose (i.e., functional and non-functional), and test automation
T14	Acceptance testing	Presents techniques to plan, execute and analyse testing with final users. Teams are oriented to execute such techniques in their projects.
T15	Interface Testing	Introduces tools to execute GUI testing for web systems, e.g., Selenium.
T16	Version controlling	Trains students to understand and apply concepts of software version control through practical activities in classroom using Git. Teams are requested to use GitHub for version control in their projects.
T17	Configuration management	Familiarises students with main concepts of configuration management topics, i.e., configuration itens, traceability, version control, release, software building. Change and release management concepts and processes also are studied;
T18	Data protection	Introduces the Brazilian's personal data protection regulation.

programming languages were JavaScript, Python, Java, and C. Almost all students have had experience using DBMS, mostly MySQL and PostgreSQL. Knowledge and experience with git/GitHub and

Table 2: Course runs numbers and pre-course students' background

Year	Program	# Students	# Groups	Group Size	Industry Exp.	Top Language	DBMS Exp.	Unit Testing	Git/Github	Agile Methods
2020	Computer Engineering	27	4	6-7	30%	C (74%) Java (74%)	74%	Little or no experience (78%)	Little or no experience (52%)	Some experience (74%)
2021	Information Systems	34	7	4-6	50%	JavaScript (65%) Java (56%)	100%	Little or no experience (86%)	Some or good experience (70%)	Some experience (68%)
2022		31	6	5-7	58%	JavaScript (74%) Python (74%)	100%	Little or no experience (75%)	Some or good experience (68%)	Some or good experience (65%)
2023		34	6	5-7	47%	JavaScript (62%) Python (62%)	100%	Little or no experience (53%)	Some or good experience (59%)	Some or good experience (53%)

agile methods were somewhat present among most students, but Unit testing was clearly a weak point in all classes.

4.1 First Course Run - 2020

The first run of the course *Software Engineering* took place at the first semester of 2020, starting as a regular in-person course but forced into ERT due to the Covid-19 outbreak. This course syllabus corresponds to an introductory software engineering course.

Theory Content: Topics were lectured in the first two months using video lessons prepared by the teacher. For this course, topics related to end-user interaction were not covered (i.e., T1, T7, and T14 in **Table 1**) since the practical project was planned to not depend on external participants. Moreover, considering this was a basic software engineering course, some advanced topics like Software Architecture for the Web (T10), Interface Testing (T15), Configuration Management (T17), and Data Protection (T18) were not addressed. **Figure 1** depicts the list of the 11 (out 18) topics taught in this first course.

Activities: During the first two months, and as a complement for lecturers, flipped classroom-based activities were used, aiming for more autonomy for students learning the contents. An activity based on a real case was prepared for the software architecture topic, presenting the rationale behind the architectural decisions of migrating a monolith to a microservices architecture. Practical laboratory activities were proposed for version control and software testing topics, i.e., using *git* to practice version control commands and *junit* to practice unit testing. Online sessions were held using Google Meet to answer students' questions. Additionally, asynchronous communication between the teacher and students was possible using Slack.

Project: The practical project involved developing a mobile application. Students were divided into four teams with a size of six to seven members. All teams worked on the same app, but each was responsible for developing one module. At the end of the project, all modules were integrated to construct the final app. The teacher prepared a list of functional requirements and an initial version of the architecture with modules description and database model. Each team member performs one of the following roles: front-end developer, back-end developer, software tester, project manager, or integration engineer. The project was executed during the last two months. Teams planned activities to develop the module in two sprints, each lasting four weeks. Each team had one online meeting with the teacher each week. The teacher and all teams met once weekly to align their progress. At the end of each sprint, a sprint review was made for each team.

Project Assessment: The project grade was calculated as the arithmetic mean of the two deliveries. Assessment criteria included project team presentation and discussions during sprint review meetings and quality of artifacts (i.e., database and architecture conformance, requirements, architecture and code documentation, code clearness, unit tests code, running module without errors, project management, and version control). Moreover, as part of the project grade, each student went through peer evaluation by his/her team members to assess soft skills (e.g., communication, pro-activity, commitment, ethics, etc.) during teamwork.

Students Assessment: Two individual tests were applied to students. Tests assessed students' knowledge of software engineering topics and their application during software development projects. The first test was designed to be an individual article written by each study, containing a dissertation on software engineering topics taught in the course. The second test was an online form with multiple selection questions. The final grade was calculated as the weighted average of project (45%), two tests (15% each), and exercises (25% in total).

4.1.1 *First instance issues.*

Issue #1: Most students did not have expertise with the technology stack chosen for the project. Since the project started in the second part of the course, students did not have enough time to overcome the technology learning curve. This caused some frustration in students because they wanted to learn the technologies in depth. This challenge is also shared by Souza et al. [15], Suciu et al. [18].

Issue #2: Teams had difficulties with internal work alignment, which, in SCRUM, is made in a five-minute daily meeting. They could not manage the scheduling of this activity outside of class without the teacher's intervention. These difficulties are more common in distance courses and were also reported by Flach and Feitosa [6], Suciu et al. [18].

Issue #3: Working on the same application was demotivating for the teams. From the students' perspective, each team working on one different module of the same app increased the difficulty of the project, mainly due to the excess of communication required to integrate all modules. Fioravanti et al. [5] also reported such difficulties in integrating parts of the projects produced by each team.

Issue #4: Two months for the project was not considered enough time to apply all topics addressed in an introductory software engineering course.

Issue #5: Despite the students' option for recorded lectures instead of synchronous classes, at the end of the course, they felt it was not a good idea because of the difficulty of scheduling time, establishing a routine for watching classes, and maintaining focus. All of this was a demotivating factor for their studies.

Issue #6: The two tests designed for this course were overwhelming, considering the online format and the high demand for time to project development.

4.2 Second Course Run - 2021

The second run occurred in the first semester of 2021 under similar conditions imposed by the pandemic and ERT. Unlike the first run, this experience occurred during the advanced course *Software Development in Information Systems*, offered at University B for Information System students. However, based on experience from the first run, some changes were proposed, i.e., teams started the practical project in the first week of the semester, teams had direct contact with the final users and clients, and no individual test was planned. Moreover, executing two additional sprints during the practical project was possible.

Theory Contents: Based on the class students' characterization, some changes were made to the syllabus. Theoretical content focused on eleven software engineering topics as presented in **Figure 1**. Topics such as design thinking (T1), interviews to requirements elicitation (T7), and user acceptance testing (T14), were introduced as we expected students to interact with stakeholders such as end users. Software architecture for the web (T10) and data protection (T18) were also part of the syllabus to introduce students to advanced topics. Additionally, students were presented with topics such as agile software development models (T3), agile project management (T5), requirements engineering (T6), software testing (T13), and version control (T16), aiming to refresh their knowledge and facilitate its application in the practical project. Finally, since this is an advanced software engineering course, some topics (i.e., T2, T4, T8, T9, T12, T15, T17) were not considered because they were taught in previous program courses.

To overcome one of the issues identified in the previous run, classes were given synchronously using Google Meet once a week. All classes were recorded and made available to students.

Activities: Considering the virtual format, most activities were proposed following the flipped classroom method, in which course material was available as online materials, comprising short videos, slides, notes, and links to supplementary reading. Students are required to cover this material before class. In addition to flipped classes, two software industry professionals (a scrum master and a software tester) were invited to present and discuss their experiences on certain topics taught during the course. Considering the students' lack of knowledge of software testing, an individual lab activity covering the planning, coding, and execution of unit and integration test cases using automation tools was conducted.

Project: Unlike the first course, the students had to develop their project from scratch from the beginning of the run. During the first class, the teacher established deadlines and a series of deliverables for each of the four planned sprints. Each sprint lasted four weeks. To support the students, the professor had bi-weekly meetings since the beginning with each group to assess their progress and provide guidance/feedback. At the end of a sprint, all groups had to make a presentation addressing the deliverables developed for the whole class. Finally, a sprint retrospective is performed to discuss issues and lessons learned. The backlog for the next sprint is planned as well.

The requested deliverables for each sprint are related to the theoretical content addressed before during classes and activities, providing a basis for the groups to develop the project. The deliverables requested for the **first sprint** were: (i) a document explaining the project idea and stakeholders' needs; (ii) mock-ups or some initial prototype validated by a stakeholder; (iii) the planning and results of interviews with end users; (iv) a requirements document containing use cases or user stories; (v) a preliminary product backlog; (vi) a project management tool configured for sprints with group meeting schedules, time estimated and students assigned for each project activity; (vii) the technology stack selected and its rationale. For **second sprint**, teams had to present: (i) sprint backlog time estimates and assign tasks to group members; (ii) implementation of estimated user stories; (iii) a software architecture description; (iv) a test cases plan; and (v) the code for test cases. Deliverables of **third** and **fourth** sprints included (i) all documentation developed so far and (ii) any new source code developed and versioned on GitHub. Finally, the groups had to (i) deploy the product in production and (ii) provide the planning and results of end-user acceptance tests for the **fourth** and last sprint. Sprint review reports and the estimates for the next sprint were also requested for the 2nd, 3rd, and 4th sprints.

In the 2021 course, teams were asked to interact with end users during the first and final sprints to validate the idea, elicit requirements, and validate the product increment in operation.

Students Assessment: No individual tests were designed for this course instance; therefore, the course's final grade was determined by the weighted average of the four project deliveries and lab activities. The project weighed 95% of the final grade, and activities weighed the remaining 5%. To assess the project, in each sprint review, the teacher assessed teams regarding the quality of deliverables, completeness and accuracy of the assets, accomplishment of estimates, teamwork, and milestones achievement. Sprint reviews had incremental weights, i.e., 15%, 20%, 30%, and 35%, respectively, to deliveries in the first, second, third, and fourth sprints.

4.2.1 *Second instance issues.*

Issue #7: During the sprint review meetings, the teacher perceived some students lacked a deep comprehension of theory concepts taught in the course. It was noticed that, for some students, studying the theoretical contents of the course was not a priority because no individual test would be applied. This situation reinforces the challenge of managing a dual-focus software engineering course to strengthen students' theoretical knowledge and practical skills, as previously reported in Soska et al. [14].

Issue #8: During the Covid-19 pandemic, as an exceptional measure, the university made it possible for students to enroll in more courses than those possible in a face-to-face format. 56% of students were enrolled in between 6 and 11 different courses motivated by the idea they could graduate earlier. However, this scenario made it difficult for students to organize their time and conciliate their studies with other activities such as internships (50% of students) and capstone projects (27% of students). Nevertheless, all students reported they felt motivated during this course's project development. Students' time management has been a constant challenge in PjBL-oriented software engineering courses, as reported by Flach and Feitosa [6], Pérez and Rubio [12], Souza et al. [15].

Issue #9: Despite the teams having used Discord and Google Meet tools to enhance their communication at a distance, most students reported difficulties interacting with their colleagues and synchronizing their work. They felt face-to-face communication would work better. Internal team communication is a challenging soft skill when working with PjBL, as highlighted by Flach and Feitosa [6], Suciu et al. [18].

4.3 Third and Fourth Course Runs - 2022 and 2023

Due to the improvement in the pandemic situation and the end of ERT, both last runs took place in an in-person format. These runs were also offered by University B during the first semester of the corresponding year and shared a similar design as depicted in **Figure 1**. Some slight changes compared with the 2021 run were applied.

Theory Contents: The syllabus of the 2022 course included a new topic, i.e., T8 - product quality, in which the ISO/IEC 25010 model's quality attributes definitions were studied. Classes were taught face-to-face once a week. This topic was not required in the 2023 instance since students already knew it from previous courses. Instead of this, in 2023, two topics were added: T12—Architectural Decision Records (ADRs) and T15—Interface Testing.

Activities: Most activities in 2022 and 2023 were practical labs in the classroom. The labs were proposed for topics such as product idea specification, quality attribute scenarios, user story estimates using story points, architectural decision records (ADRs), software architecture representation with C4, and unitary, structural, interface, and acceptance testing. Specifically for the 2022 course, one seminar was planned with an AWS architect and team leader. During the 2023 course, a real case-based activity was designed to teach the main software architecture concepts. The case studied involved a robot's architecture and source code.

Project: The project design for the 2022 course was similar to the one executed in 2021. Few changes were proposed. Three sprints were planned instead of four so students could have more time for each sprint. Teams also interacted with final users in the first and last sprints. Meetings between the teacher and each team happened twice monthly. The Discord application was used to support meetings outside the normal class schedule.

To avoid overloading some students, the project design for the 2023 run was slightly different. The professor strongly suggested that all team members act as developers or testers and also select one of the roles of scrum master, product owner, architect, or operations engineer.

Course Assessment: In both years, one final test was proposed to encourage each student to learn the theory so they could contribute better to the project instead of relying on colleagues' knowledge. The course's final grade was given by the weighted average of the individual test (15%), lab activities (15%), and the three project deliveries (70%), which had an incremental weight by sprint (i.e., 20%, 30%, and 50%).

4.3.1 *Third and fourth instance issues.*

Issue #10: In the 2022 course instance, each student chooses one software engineer profile to participate in the practical project. Each team comprised one product owner, one scrum master, one software architect, and various front-end and back-end developers and testers. Depending on the size of each team, students executed one or various roles. Some teams reported an overload of students with developer roles. In that case, it was observed that students with roles as product owner and scrum master did not help with coding and testing activities. Few teams were more proactive and divided development and testing work in a more balanced way.

Issue #11: Also, in 2022, some students had an initial idea of the software product they desired to work on as the project course. The professor allowed them to create teams to work on those ideas during the course. However, it was observed the students who "owned" the ideas limited the creative work of their colleagues, and the other students did not feel confident in proposing changes or making decisions during the project execution.

Issue #12: In the 2023 course, in some teams, during role selection, students with industry experience in a certain role (e.g., back-end or front-end developer) tend to perform the same role in the project. It was perceived that, by doing this, most of the workload of such a role falls on the more experienced since, in that way and from the teams' perspective, they could produce more code. This scenario leads to an overload for experienced students. Moreover, the not-so-experienced students act as observers who are afraid of messing up the code produced by their colleagues.

Those issues corroborate findings of Souza et al. [15], Suciu et al. [18]. In both studies, teachers reported that some team members lack commitment to the team and engagement with the project. Therefore, work distribution between team members is unbalanced, overwhelming the more proactive or experienced students.

5 RESULTS

As mentioned in Section 3, two forms were used to capture students' opinions about the course: A pre-course form aimed to understand students' expectations, previous knowledge and experience with software development in practice. The post-course form intended to obtain, from students' perspective, information about how well the course was conducted and whether it was helpful for their learning paths. No personal information was collected in both forms, and students accepted all data treatment terms. Students' participation in filling out the forms was voluntary and did not affect their grades or assessments. The remainder of this section presents the analysis results of students' answers and the teams' project grades.

5.1 Students' expectations and suggestions

Students were asked about their expectations of the course. 77.8% (98/126) of students answered this question. As a result, 25.3% (32/98) looked to develop a product using current methods and technologies used in the software industry; 15,1% (19/98) desired to gain practical experience with technologies for front-end, back-end, and DevOps; 12% (15/98) desired to learn more about software engineering to help their insertion in the job market; 10,3% (13/98) wanted to improve soft-skills such as teamwork and time organization; 8,7% (11/98) awaited to gain practice with agile projects; and 3,2% (4/98) hoped to discover new technologies, methods, and concepts in software engineering. An unexpected finding was that 3,17% (4/98) of students did not have any expectations about the course because they did not intend to act professionally in the software industry. No significant difference was found between students' expectations in online and in-person courses.

Moreover, students were requested to provide suggestions about strategies they prefer to follow in the course. Some differences in suggestions between online and in-person class students were found. Suggestions from 28 out of 61 students in online courses who answered this question were related to (i) have more autonomy to form work teams (7/28); (ii) estimate feasible delivery deadlines considering the division of their time with other activities (e.g., scientific initiation projects, internships, full-time work in software companies, other courses, and other academic activities) (6/28); (iii) provide video lessons (3/28); (iv) have individual assessments in addition to the practical project (2/28); and (iv) encourage groups working in a synchronous way supervised by the teacher (2/28). From the 39 out of 63 students on in-person courses, the hints were concerned with (i) encouraging groups to work in a synchronous way supervised by the teacher (8/39), (ii) planning a project that does not overload students (8/39); (iii) providing practical exercises in addition to the practical project (6/39); (iv) estimating feasible delivery deadlines (4/39); (v) synchronously lecturing theory classes; and (vi) following by close the teams projects evolution (3/39).

5.2 Project grades

For the first course run in 2020, the project had two deliveries, each with the same weight. Student groups achieve an average project grade of 8.2 (median = 8.3, std = 0.5, min = 7.6, and max = 8.8). As depicted in the top-left hand of **Figure 2**, all groups in the 2020 class improved their project grades in the second delivery. The green bars show how much a group's grade improved between deliveries.

The project grade average for the 2021 class was 8.3 (median = 8.4, std = 0.4, min = 7.6, and max = 8.6), a result that was quite similar to last year's run. It is possible to notice in the top-right hand of **Figure 2** that five out of seven groups had at least one grade drop in deliveries 2 to 4, highlighted as a red color bar. Only two groups (i.e., G2 and G3) continuously improved their deliveries, increasing grades.

For the 2022 course, the project grade average was 8.6 (median = 8.7, std = 1.2, min = 7.5, and max = 9.8). A slight improvement in final project grades was noticed compared to the two previous runs. As illustrated at the bottom left hand of **Figure 2**, five out of six groups had a grade decline in at least one of the project deliveries. Only one group (i.e., G1) had continuous improvement in delivery grades.

During the last course run, in 2023, the project grade average for this class was 8.4 (median = 8.4, std = 1.1, min = 6.5, and max = 9.6). The amount and weight of project deliveries were the same as in the 2022 course. Similarly to the previous course run, in 2023, only one group (i.e., G5) did not have a grade drop in the second and third project deliveries.

Overall, no relevant difference was found between the groups' project grades considering the course modality, i.e., totally virtual or in-person. The quality and completeness of project deliveries were quite similar in all courses, with few groups as exceptions, i.e., G1 in 2022 with a high project grade of 9.7 and continuous delivery improvement, and G4 in 2021 and G2 in 2022 with low project grades (i.e., 7.6 and 7.5, respectively) and constant delivery decreasing).

Grades were perceived to decrease mainly in intermediate project deliveries, i.e., at the end of the second sprint. This result can be explained since, in this sprint, groups start working with technologies, which implies investing efforts to learn such technologies while at the same time understanding the team dynamics. Despite being trained in time and effort estimates to select the workload to be executed in a sprint considering their academic and professional reality, most groups had difficulties organizing their time to conduct project activities, leading, in some cases, to a decrease in the last project deliveries grades.

5.3 Challenges faced by students during projects

Only 48% (60/126) of students in all courses answered the post-course form. This form was intended to identify difficulties the students faced during the course in both modalities, virtual and in-person.

Most students (93%) reported the as the main difficulty as reconciling project execution with other academic and professional activities. A second difficulty, reported by 65% of the students, was teamwork, mainly related to effective communication, team members' commitment, and work synchronization. The third most reported issue, by 50% of respondents, was not knowing the technologies chosen by the team. Additionally, online students reported specific difficulties conducting the project in a remote environment (12.2%), mainly due to the lack of face-to-face meetings to decide relevant actions for the project development. These findings are similar to the students' challenges reported in recent experiences of Pérez and Rubio [12], Suciu et al. [18].

5.4 Most difficult and liked topics

Regarding the theory contents, 50% of students considered software architecture the most difficult topic to understand, followed by automated software testing (38,3%), software product quality (21,7%), and software time and effort estimates (21,7%). In contrast, students reported software architecture (60%), agile methodologies (58.33%), product quality (55%), software testing (45%), and software time and effort estimates (40%) as the topics they liked the most. Even though some of those topics were the most difficult to understand, they were applied successfully to the practical project, evidenced by the quality of the group's deliverables. Similar results were obtained by Souza et al. [15] who found that students in PjBL courses positively perceive the project contribution on learning more difficult topics such as software design (architecture) and agile methodologies.

5.5 Students' comments and suggestions

Students could openly write general opinions about the course execution or suggestions for course improvements in the future.

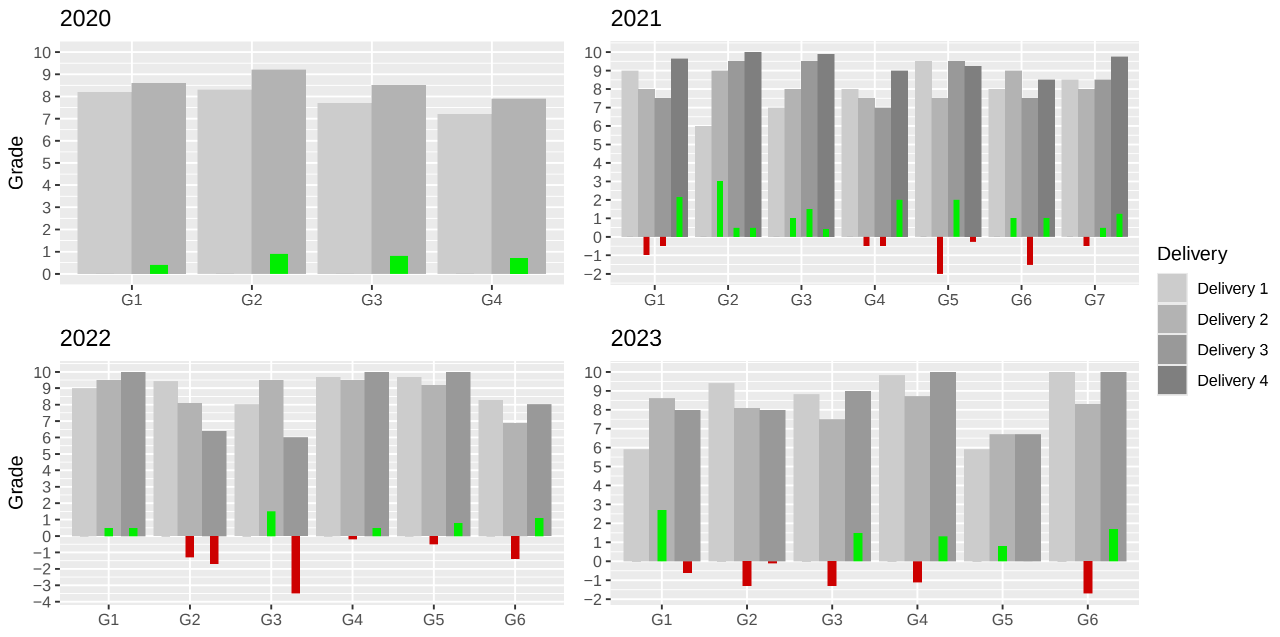

Figure 2: Delivery grades for groups by course. Green and red bars represent the grade difference (positive and negative, respectively) between the current and previous delivery. Grades range is from 0 to 10, with 6 as the minimum pass mark.

Generally, the students liked the course because they understood better how all learned knowledge in other courses (e.g., database, web development, software engineering topics, UX/UI, etc.) fit together. Examples of students comments are:

- *"I really liked the subject; I think it encouraged me to improve in some of the areas I learned"*;
- *"It is a very interesting subject. It uses a lot of the knowledge acquired during the undergraduate course and expects maturity from the student"*;
- *"I really liked the discipline and the way it was conducted. At first, I thought there would be a time requirement far beyond what I had available to dedicate. However, the entire semester course was coherent and very productive, covering a practical compilation of practically all of our undergraduate courses. Therefore, I consider it perhaps the most complete and important subject in the course. I really liked it!"*;
- *"It was a productive course"*;
- *"My experience with the discipline was very satisfactory. I really liked the approach"*.

Two students who had industry experience of more than 2 years commented about the realism of the course with software industry practices, corroborating similar results found by Souza et al. [15], Ståhl et al. [17]:

- *"The way the team worked is very similar to the way things are done in my company"*.;
- *"I see many techniques that we practice in the course in my company"*.

Other students suggested improvements to the course and project execution. For instance, one student felt important that the teacher *"orients how to deal and communicate with someone acting with less responsibility in the team"*. In a similar perspective, another student suggested a *"closer monitoring of the teacher to reconcile internal team problems, such as overload or lack of responsibility of members"*. Finally, five students suggested including CI/CD (Continuous Integration and Deployment) or DevOps topics to help teams choose cloud infrastructure and release workflows during software deployments in preparation for end user tests.

5.6 Summary of Course Evolution

Executed teaching strategies were motivated by issues identified in previous course runs, allowing a continuous course evolution and improving the PjBL application. Table 3 relates the issues that motivated course changes in next course iterations, the result (e.g., not satisfactory, almost satisfactory, or satisfactory) of applying such changes, and recommendations to maintain or not a change in next course offering. More maturity and stability were perceived in the 2023 course, although there are still open challenges to be addressed in the near future.

6 LESSONS LEARNED

In this section, the authors present their most relevant lessons learned through four years of teaching software engineering with PjBL.

Lesson #1: Start the course knowing your students' background. An important activity in setting up the course is understanding students' previous knowledge. This lesson is also shared by Flach and Feitosa [6], in which it is recommended to survey students' previous experience and knowledge to adjust the course plan if required.

Table 3: Rationale behind course evolution

2021 Iteration			
Issues from previous iteration	**Changes adopted in current iteration**	**Results of changes**	**Recommendation for the next iteration**
#1 - High technology learning curve	To start technology studies in the first week	Not satisfactory	Maintain
#2 - Teams internal work alignment	Synchronous team meeting by week.	Not satisfactory	Change
#3 - Lack of motivation by working on a project defined by the teacher	Teams choose their projects and interact with final users and clients	Satisfactory	Change
#4 - Few time to the project	To start the project in the first week. Project with four sprints, each sprint with a three weeks window.	Not satisfactory	Start to work in the project in the first week. Project with three sprints. Each sprint during four weeks.
#5 - Recorded lectures demotivated studies	Synchronous encounters with the teacher	Not satisfactory	In-person encounters with flipped activities
#6 - Two individual tests was overwhelming	No individual test. Only the team project	Not satisfactory	Individual tests are required.
2022 Iteration			
Issues from previous iteration	**Changes adopted in current iteration**	**Results of changes**	**Recommendation for the next iteration**
#7 - Low theoretical domain by students	One individual test and the team project	Satisfactory	Maintain
#8 - Teams time management	More focus on realistic project time estimates. Closer follow-up by the teacher.	Almost satisfactory	Add calculation of error metrics for estimates
#9 - Teams internal work alignment	Use discord channels for the teacher to check the weekly team meetings	Almost satisfactory	Add in-person work allignment during class
2023 Iteration			
Issues from previous iteration	**Changes adopted in current iteration**	**Results of changes**	**Recommendation for the next iteration**
#10 - Roles distribution among team members is unbalanced	Each student with two roles: developer and one between the following options PO, SM, soft. architect, tester, UI/UX designer, Ops engineer.	Almost satisfactory	Maintain. Consider improve groups formation.
#12 - Experienced students centralize most of the development			
#11 - Idea's owner centralize most of the development	Idea defined through team members discussions and voting.	Satisfactory	Maintain

In our experience, the pre-course form was essential to have an idea about the theory contents to be reinforced or introduced and the technical capacities of students to be explored during the project. However, the teacher must know that answers to this kind of questionnaire are informative and not confirmatory. In some cases, it is required to confirm actual knowledge of essential topics that can impact project development. In our course, an example of such a topic is agile methodologies. In the 2023 course, more than 50% of the students answered that they have good knowledge and practical experience in following agile methodologies, particularly SCRUM. To confirm such knowledge, the teacher conducted a formative assessment. Surprisingly, less than 20% of the students passed this assessment.

Therefore, it is important to have an overview of the student's background before starting the course; however, confirmatory assessments can be necessary for relevant topics to the project. In cases where students lack such knowledge, the teacher can design instructions for reinforcement purposes.

Lesson #2: Start working with technologies as soon as possible. The technology learning curve must be overcome before the start of the second sprint. For that, the teacher must encourage teams to select and train technologies during the first weeks of the course. In addition, class time could be destined for students to train the technology chosen for their project.

Lesson #3: Start the course with a defined team meeting schedule. Student groups had difficulties handling diverse academic and professional activities and communication between members. At the course start, the teacher could define, together with each team, standard weekly time slots for team activities alignment meetings using communication applications, i.e., discord, in which the teacher can know student assistance at such meetings. Sprint review and retrospective encounters must also be defined for all teams in the first week of the course, which helps teams establish deadlines for planning their project deliveries.

Lesson #4: Encourage teams' autonomy and accountability. Students felt more motivated in the course when they could choose their project topics and the technologies they wanted to apply. The teacher can give tips about choosing the technology stack, considering characteristics such as good documentation, community size, and open training material. Group discussions to reach a consensus on selecting the technologies and project topics is a valuable activity for training soft skills such as conflict resolution.

Responsibilities and accountability for each team member must be clear from the project beginning. Students must select their roles, knowing which project artifacts (e.g., documents, models, backlogs, code, repositories, infrastructure, etc.) and sprint activities are under their responsibility. Moreover, the accountability mindset must be trained since all team members must be accountable for a project release's success or failure.

This lesson was also reported by Simpson and Storer [13]. In their report, confirmed by our experience, they stated the students must see role-specific responsibilities as their primary responsibility and be aware that they must be accountable for the quality of the outputs they produce during the project.

Lesson #5: Teach and practice with reality. Students manifested interest in software engineering topics taught with realism. For instance, using cases explaining software architecture construction and application in real industry software projects made the software architecture topic the most interesting topic to students. Similarly, talks with specialist scrum masters and project managers raise students' interest in topics such as agile actual practice and project estimates in the software industry. From the same perspective, working with real problems and real users during software project development increases the motivation of students to apply software engineering methods and techniques correctly. A similar lesson was also reported in Simpson and Storer [13].

Lesson #6: Be present for the teams, no matter the course format. As mentioned, groups must be autonomous to make important project decisions. However, in addition to theory classes, there are activities in which the teacher must intervene, namely, review of the interview plan and questions during requirements elicitation with users, the definition of the project scope, review of sprint backlog at the beginning of each sprint, quality of documentation delivered at the end of each sprint, encourage the execution of sprint retrospective and review of the planning of acceptation tests by final users in the last sprint. Additionally, proactive intervention by the teacher is relevant when conflicts between team members occur, especially in cases of overloading or low participation of team members. This lesson corroborates hints in related experiences reported by Fioravanti et al. [5], Suciu et al. [18], and Delgado et al. [2].

In our practice, we identified the importance of maintaining an asynchronous communication channel, different from e-mail or forum panels, that allows the teacher to answer team doubts quickly. We started using Slack in 2020 and moved to Discord in 2021 to 2023 courses since the last one suited better our needs. The teacher could use this channel to observe team members' interaction (or lack thereof) and identify and mediate possible conflicts. The need to define effective communication channels early in the course was also recommended by Flach and Feitosa [6].

This lesson is pertinent in any course modality, e.g., totally virtual, hybrid, or in-person.

Lesson #7: Maintain individual students' assessments. From the author's experience, assessing individual students' knowledge of software engineering applied to the project is relevant. Ideally, each student can conclude the course by understanding software engineering practice beyond the specific role he/she has performed in the project; therefore, the student has first personal experience on which to base his/her professional profile.

Lesson #8: Leverage past experiences to anticipate and address challenges: Software engineering course syllabuses evolve as the industry evolves. Teaching in software engineering is now more challenging than a decade ago since society and the software industry are changing continuously. From our experience, adapting course syllabus and adopting new teaching methodologies in the classroom, as suggested by academic societies, was overwhelming and required us to leave our comfort zones. By the time of our first course, we had used some insights from fellow teachers, helping us a lot but not enough to prepare us for all the challenges we faced. Good software engineering teaching reports, such as those introduced in **Section 2**, are rich know-how sources for beginning teachers in PjBL since they summarize the benefits and drawbacks of applying specific teaching strategies in the classroom.

7 CONCLUSION

Computing curriculum by ACM[1] and SBC[2] suggest considering a project course as an essential approach to give students the opportunity to solve challenging projects in which them can deepen their knowledge in a transversal way. Therefore, an increasing employment of PjBL in software engineering education and training is expected.

Using PjBL demands more effort and time from instructors for planning, executing, and learning compared with traditional classes (i.e., lecturers and tests) and with other active learning approaches (e.g., flipped classroom or case-based learning) [7, 9]. Moreover, PjBL courses require continuous improvement to adapt them to students and industry realities and in a sustainable manner [8, 9, 17].

Improving a course is a trial-and-error task that requires years of teaching. Teachers who desire to adopt PjBL in their courses could benefit from lessons learned by their peers, thus decreasing their learning curve and avoiding common issues found when this teaching method is implemented. The research of Cico et al. [1] evidenced that PjBL is the learning method most used in software engineering courses. However, by 2021, less than 10% of software engineering education research reports challenges and lessons learned in applying this method.

This work presented four years of accumulated experience in teaching software engineering with PjBL in both virtual and face-to-face modalities. Such experience allowed the course to improve over the years. Issues found after each course run were presented, as well as lessons learned to be applied in future course offerings. We expect this report to give our colleagues insights into what actions to avoid and which ones to consider in their PjBL-based software engineering courses.

From our teaching practice, we did not identify a significant difference between conducting a virtual or in-person course with PjBL; therefore, we believe all challenges and lessons reported herein are valid for both formats. Regardless of the course modality, open challenges remain, especially those related to teamwork organization and team members' communication and engagement.

The team formation and role allocation between team members strategies should be reconsidered to allow more balanced teamwork. Until now, in our course, students chose their own teams. While it seems this strategy leads to slightly better performance in teams [10], it did not mitigate internal teamwork problems, e.g., lack of team member engagement and overwhelming of compromised students. Therefore, it is important to investigate different ways of forming teams and assessing individual soft skills whilst maintaining team performance. By now, in our courses, assessments are related to the quality of artifacts, outputs, releases, and correctness of software engineering techniques execution. For new courses, it is necessary to include evaluation criteria that allow the assessment and improvement of students' individual soft skills.

Regarding syllabus changes, special attention will be given to topics such as DevOps, security, and refactoring, with the aim of increasing the quality of software produced by the students during the project.

8 AVAILABILITY OF ARTIFACTS

Artifacts related to data collection and analysis used in this study are available in [11].

ACKNOWLEDGMENTS

To all our students who believed in us. To the "Pró-Reitoria de Pesquisa e Inovação (PRPI), Universidade de São Paulo" (Process: 22.1.09345.01.2).

REFERENCES

[1] Orges Cico, Letizia Jaccheri, Anh Nguyen-Duc, and He Zhang. 2021. Exploring the intersection between software industry and Software Engineering education - A systematic mapping of Software Engineering Trends. *Journal of Systems and Software* 172 (2021), 110736. https://doi.org/10.1016/j.jss.2020.110736

[2] David Delgado, Alejandro Velasco, Jairo Aponte, and Andrian Marcus. 2017. Evolving a Project-Based Software Engineering Course: A Case Study. In *2017 IEEE 30th Conference on Software Engineering Education and Training (CSEE&T)*. 77–86. https://doi.org/10.1109/CSEET.2017.22

[1]https://www.acm.org/education/curricula-recommendations

[2]https://www.sbc.org.br/documentos-da-sbc/category/131-curriculos-de-referencia

[3] Vinicius Gomes Ferreira and Edna Dias Canedo. 2020. A Design Sprint based model for User Experience concern in project-based learning software development. In *IEEE Frontiers in Education Conference (FIE)* (Uppsala, Sweden, 2020-10-21). IEEE, 1–9. https://doi.org/10.1109/FIE44824.2020.9274214

[4] Maria Lydia Fioravanti, Bruna Oliveira Romeiro, Leo Natan Paschoal, Brauner Oliveira, Simone R. S. De Souza, Ellen Francine Barbosa, and Ana M. Moreno. 2023. Software Engineering Education Through Experiential Learning for Fostering Soft Skills. In *2023 IEEE Frontiers in Education Conference (FIE)* (2023). IEEE, College Station, TX, USA, 1–8. https://doi.org/10.1109/fie58773.2023.10343452

[5] Maria Lydia Fioravanti, Bruno Sena, Leo Natan Paschoal, Laíza R. Silva, Ana P. Allian, Elisa Y. Nakagawa, Simone R.S. Souza, Seiji Isotani, and Ellen F. Barbosa. 2018. Integrating Project Based Learning and Project Management for Software Engineering Teaching: An Experience Report. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education* (Baltimore Maryland USA, 2018-02-21). ACM, 806–811. https://doi.org/10.1145/3159450.3159599

[6] Christina Von Flach and Daniela Soares Feitosa. 2023. Teaching and Promoting Engagement with OSS: Yet Another Experience Report. In *Proceedings of the XXXVII Brazilian Symposium on Software Engineering* (<conf-loc>, <city>Campo Grande</city>, <country>Brazil</country>, </conf-loc>) *(SBES '23)*. Association for Computing Machinery, New York, NY, USA, 534–543. https://doi.org/10.1145/3613372.3614190

[7] Chetna Gupta. 2022. The Impact and Measurement of Today's Learning Technologies in Teaching Software Engineering Course Using Design-Based Learning and Project-Based Learning. *IEEE Transactions on Education* 65, 4 (2022), 703–712. https://doi.org/10.1109/TE.2022.3169532

[8] Philippe Kruchten. 2011. Experience teaching software project management in both industrial and academic settings. In *2011 24th IEEE-CS Conference on Software Engineering Education and Training (CSEE&T)*. IEEE, Waikiki, Honolulu, Hawaii, 199–208. https://doi.org/10.1109/CSEET.2011.5876087

[9] Stephan Krusche and Lukas Alperowitz. 2014. Introduction of continuous delivery in multi-customer project courses. In *Companion Proceedings of the 36th International Conference on Software Engineering* (Hyderabad, India) *(ICSE Companion 2014)*. Association for Computing Machinery, New York, NY, USA, 335–343. https://doi.org/10.1145/2591062.2591163

[10] Henrik Hillestad Løvold, Yngve Lindsjørn, and Viktoria Stray. 2020. Forming and Assessing Student Teams in Software Engineering Courses. In *Agile Processes in Software Engineering and Extreme Programming – Workshops*, Maria Paasivaara and Philippe Kruchten (Eds.). Springer International Publishing, Cham, 298–306.

[11] B. Oliveira and L.' Garcés. 2024. Supplementary material for the work intitled: Teaching Software Engineering with Project-Based Learning: A Four Years Experience Report. on-line. https://doi.org/10.5281/zenodo.12794894

[12] Beatriz Pérez and Ángel L. Rubio. 2020. A Project-Based Learning Approach for Enhancing Learning Skills and Motivation in Software Engineering. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education* (Portland OR USA, 2020-02-26). ACM, Portland, OR, USA, 309–315. https://doi.org/10.1145/3328778.3366891

[13] 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

[14] Alexander Soska, Irmgard Schroll-Decker, and Jürgen Mottok. 2014. Implementation of practical exercises in software engineering education to improve the acquirement of functional and non-functional competences: A field report about project-based learning in software engineering. In *2014 International Conference on Interactive Collaborative Learning (ICL)*. 338–345. https://doi.org/10.1109/ICL.2014.7017795

[15] Maurício Souza, Renata Moreira, and Eduardo Figueiredo. 2019. Students Perception on the use of Project-Based Learning in Software Engineering Education. In *Proceedings of the XXXIII Brazilian Symposium on Software Engineering (SBES)* (Salvador Brazil, 2019-09-23). ACM, Salvador, BA, Brazil, 537–546. https://doi.org/10.1145/3350768.3352457

[16] A Strobel, J. & van Barneveld. 2009. When is PBL More Effective? A Meta-synthesis of Meta-analyses Comparing PBL to Conventional Classrooms. *Interdisciplinary Journal of Problem-Based Learning* 3(1) (2009), 44–58. https://doi.org/10.7771/1541-5015.1046

[17] Daniel Ståhl, Kristian Sandahl, and Lena Buffoni. 2022. An Eco-System Approach to Project-Based Learning in Software Engineering Education. *IEEE Transactions on Education* 65, 4 (2022), 514–523. https://doi.org/10.1109/TE.2021.3137344

[18] Dan Mircea Suciu, Simona Motogna, and Arthur-Jozsef Molnar. 2023. Transitioning a project-based course between onsite and online. An experience report. *Journal of Systems and Software* 206 (2023), 111828. https://doi.org/10.1016/j.jss.2023.111828

Received 24 May 2024; revised 25 June 2024; accepted 11 July 2024