Simulating the Software Development Life Cycle in an Educational Setting: Insights from Individual and Collaborative Student Projects

Luka Četina[1,*], Vasilka Saklamaeva[1] and Luka Pavlič[1]

[1]Faculty of Electrical Engineering and Computer Science, University of Maribor, Koroška cesta 46, Maribor, 2000, Slovenia

Abstract

A good software development education provides students with a combination of theoretical knowledge and practical experiences. However, preparing students for real-world development can be a challenge. Isolated assignments often fall short of adequately preparing students for a work setting where they will not only develop new software but also plan, improve and maintain existing systems throughout their lifecycle. Simulating the entire software development lifecycle is crucial, as it not only improves the students' skills but also better reflects the complexities of real-world software development. Furthermore, developers rarely work alone and must coordinate and communicate with their teams as well as customers. Curriculums often focus predominantly on technical knowledge and do not address the need for improving the students' collaboration and communication skills. As part of a course, we adopted an approach aimed at simulating the software development life cycle, where students planned, developed and iterated their Information System. To give the students the experience of developing in teams, while avoiding the pitfalls of group work and ensuring everyone acquired the necessary knowledge, we implemented a hybrid approach combining individual and group work. The goal of this paper is to share our approach and outline key findings and experiences.

Keywords

Software development education, software development lifecycle, agile development

1. Introduction

There is a great demand for quality software developers and the fast pace of changes in tech requires developers to constantly improve and acquire new knowledge [1]. Just as tools and technology stacks are quickly changing, so are the approaches used in the software development process. To prepare students for real-world development they not only need sufficient technical skills but also the knowledge of how to apply them at each stage of development and deliver a complete product. Understanding the different stages of software development, as defined in the software development lifecycle (SDLC), is essential for professional developers, who not only write code but also design systems and respond to changing requirements [2], [3], [4].

Despite this, many university courses still rely on isolated programming tasks or final projects that emphasize code production without simulating the full development lifecycle. As a result, students may graduate with limited experience in areas such as requirement analysis, project planning, software design, testing strategies, and deployment workflows [5]. Since most programmers work in teams, courses must also address the collaborative and social aspects of software development. Giving the students a chance to apply their skills in a collaborative development environment better prepares them for the complexities of real-world development while simultaneously improving their technical as well as collaboration skills [1], [6]. Simulating the software development lifecycle in an educational setting is a good way to bridge this gap, but can be challenging, due to organizational constraints as well as the differing level of the student's skills and commitment. One of the key challenges is ensuring that all students are fully engaged and acquire the necessary skills. While group projects offer

NWISEd 2025: Workshop on Co-Creating New Ways, of Information Systems Education, September 10–11, 2025, Maribor, Slovenia

*Corresponding author.

luka.cetina@um.si (L. Četina); vasilka.saklamaeva@um.si (V. Saklamaeva); luka.pavlic@um.si (L. Pavlič)

0009-0000-0267-1092 (L. Četina); 0009-0005-0221-4840 (V. Saklamaeva); 0000-0001-5477-4747 (L. Pavlič)

opportunities for collaboration, they often come with the risk of uneven participation. To address this, it is essential to implement mechanisms that promote individual accountability while still fostering team-based collaboration.

In our course we adopted a hybrid approach, combining both individual and group assignments to ensure that each student independently acquired the necessary technical knowledge, while still benefiting from the experience of collaborative development. The goal of this paper is to share out approach and outline key findings and experiences.

The structure of this paper is as follows. In the next chapter, we briefly review related work in software engineering education, focusing on approaches that simulate real-world practices. Chapter 3 describes the course setup and the structure of assignments as well as how the SDLC was simulated. In Chapter 4, we analyze what worked well, what could be improved and list lessons learned. Chapter 5 contains the conclusion and ideas for future work.

2. Related work

Devedžić and Milenković [5] present an extensive case study based on eight years of teaching agile software development in a university context. Their work not only describes the practical aspects of implementing agile practices in the classroom but also reflects on how students gradually adopt the principles of iterative development, collaboration, and responsiveness to change. Over the years, they observed that students often struggle when first exposed to agile, particularly with practices such as iterative planning, retrospectives, and maintaining continuous communication. However, the longitudinal nature of the study shows that with consistent exposure and appropriate guidance, students become increasingly proficient and more confident in agile development. The authors also highlight that curriculum design for agile teaching requires continuous improvement, as rigid course structures fail to capture the evolving and adaptive spirit of agile methodologies.

Schroeder et al. [6] report on the setup, execution, and outcomes of two software development lab courses with a specific focus on agile methodologies. Their study highlights how hands-on, team-based projects allow students to experience the dynamics of collaborative development in a setting that mirrors industry practices. A key contribution of their work is the detailed account of challenges that arise when academic structures—such as semester timelines and course schedules—are combined with agile practices that rely on short, iterative development cycles. These structural mismatches often require careful adjustment in order to maintain both pedagogical effectiveness and authenticity of the agile experience. In addition, they emphasize that effective collaboration cannot be assumed, but must be deliberately supported through scaffolding, guidance, and continuous feedback.

Patani et al. [7] examine the impact of using GitHub as a collaborative tool in software engineering courses, with a focus on how such platforms influence engagement and accountability. Their findings show that GitHub enhances student participation by making project progress and individual contributions transparent. The platform not only supports technical collaboration through version control but also provides a traceable record of work, which facilitates both peer-to-peer accountability and instructor assessment. Importantly, the study notes that GitHub can motivate students to contribute more consistently, as their work becomes visible to both teammates and instructors. The authors argue that such transparency reduces common problems of uneven participation in group projects and fosters a culture of shared responsibility. By integrating GitHub directly into coursework, educators are also able to bring classroom practices closer to industry standards, providing students with practical skills and habits that are directly transferable to professional settings.

Chen and Chong [8] introduce the meetings-flow approach as a structured method to sustain collaboration during senior project courses. This approach organizes communication into a sequence of structured meetings that provide checkpoints for progress monitoring, guidance, and engagement throughout the project lifecycle. Their empirical study demonstrates that this structure improves not only teamwork and productivity but also the overall quality of project outcomes, as students remain more engaged and accountable when interactions are formalized. A key insight from their work is

the importance of simulating realistic development conditions by embedding evolving requirements, structured communication, and stakeholder feedback into the educational process. They argue that providing such scaffolding ensures students gain experience not only with technical development tasks but also with the collaborative and organizational challenges of real-world projects.

Buffardi [9] proposes a method that combines Git commit logs with Kanban board user stories to assess individual contributions in software engineering projects. This approach allows instructors to align automatically captured data with peer and instructor evaluations, resulting in a more objective and reliable assessment of teamwork. One of the strengths of this method lies in its ability to make invisible aspects of collaboration visible: for example, identifying students who contribute disproportionately less or those who take on a greater share of the work. By providing a clearer picture of participation, instructors can intervene earlier to address imbalances in team effort. The author claims this approach also supports fairer grading practices, as contribution metrics are not based solely on self-reports or subjective impressions. More broadly, Buffardi's work demonstrates the value of using development artifacts as instruments for evaluation and feedback in education.

Gitinabard et al. [10] analyze GitHub logs from student programming projects to infer teamwork styles and to derive insights into how students collaborate. They categorize teamwork behaviors into patterns such as collaborative (balanced contributions across members), cooperative (division of tasks with limited overlap), and solo-submit (dominated by one contributor). These patterns are identified through commit activity, file ownership, and temporal distribution of contributions. The results demonstrate that automated log analysis can reveal hidden dynamics in student teams that might not be apparent through observation or self-reporting alone. Such insights are valuable because they allow instructors to better understand group functioning, identify at-risk teams, and design interventions to improve collaboration. Moreover, the study illustrates how large-scale log data can be leveraged to produce actionable feedback for both instructors and students.

Overall, these studies show that simulating professional development in an educational settings requires a good balance between authenticity and working within the educational constraints. Prior work has shown that agile methods, structured communication, and industry-standard tools such as GitHub can significantly enhance both technical learning and teamwork, while also introducing new challenges related to time constraints, tool adoption, and assessment. A recurring theme across the literature is that students benefit most when courses not only expose them to technical practices but also collaboration and organization. Building on these insights, the following section describes how our course was designed to simulate the software development lifecycle and address some of the challenges identified in earlier work.

3. The course structure

Our approach was implemented in the Development of Information Systems course, which is taught as part of the Bachelor's study program Information and Data Technologies at the University of Maribor, Slovenia. It lasts 15 weeks and covers topics such as requirements analysis, planning, implementation, testing and automatization. Students are split into multiple lab groups where each group has labs once per week, lasting 135 minutes. Lectures last 90 minutes per week and are performed for all students at once.

3.1. Assignments

To simulate the development of Information Systems from conception to completion, we split the work into 13 assignments. They were structured to gradually become more complex, allowing students to incrementally work on their projects while acquiring both technical and collaboration skills. The assignments were divided into individual and group tasks to ensure that each student understood the core concepts before moving on to group work. This was done to try to avoid the pitfall of group work where students do not contribute equally due to varying levels of skill. The collaborative part of the course was done in groups of 3, where students could freely choose the members of their teams.

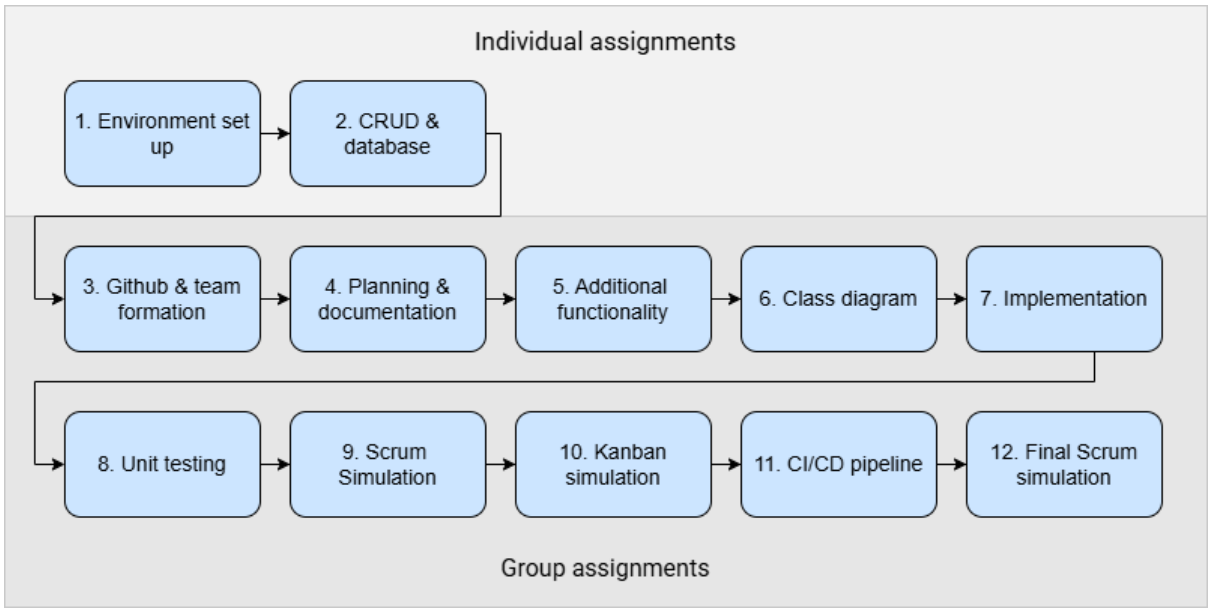

Figure 1: Structure of the assignments

Allowing students to choose their own team members was done to ensure that they worked well together and were motivated to contribute to the team. Students who did not form their own teams were assigned teams by the teaching assistants (TAs).

For some group development assignments, we chose to use agile development methodologies, namely Scrum and Kanban, as they are widely used in practice and a great fit for our team size and course duration [5]. Students were required to develop a web application. The technology stack used was: SpringBoot for backend, ReactJs for frontend and MySQL for storing data. The work from individual assignment was submitted to the online learning platform Moodle, while group assignments were uploaded to GitHub [11]. Teams showed their work to the TAs during the lab sessions, who then analyzed the project on GitHub to determine the grade for the assignment. The structure of the assignments is shown in Figure 1 and further described in the following section.

Individual assignments:

1. *Environment set up*: Each student was required to set up their working environment and create a project that works but does not yet contain any functionality. This assignment ensured that students had a working project setup and resolved any configuration issues before development began. Students could choose to implement a web application that was one of the following choices: TODO website, events website, travel planning website or a recipe website. The limited choice was implemented to ensure that the end results can be functionally comparable. They were however, not given a detailed specification but were expected to design the structure and functionality themselves within the chosen domain.
2. *Database integration and CRUD operations*: This assignment tasked students with creating an SQL database, connecting it to their project and implementing CRUD (Create, Read, Update, Delete) operations for their core functionalities. Additionally, students had to enhance their project by implementing a new feature of their choice.

The individual assignments provided each student with hands-on experience in setting up and implementing a basic application within a defined theme. Once group work began, each team selected one of their members' individual projects as the foundation for collaborative development.

Group assignments:

3. *Version control and collaboration using GitHub*: Students divided into teams of 3 and were tasked with choosing a common project topic out of the previous individual assignments. They also

needed to create their repository and necessary repository structure, as well as adding a README file, which includes a description of the project and its components, installation guide and common guidelines for developers working on the project.

4. *Project planning and documentation*: Students were required to describe their vision for the project and explain the purpose of the application, its aim and target audience. They also had to create a glossary of key terms used in their project domain and a use case diagram.
5. *Additional functionality*: Students were required to expand the use case diagram from the previous assignment and start preparing for the development of a new functionality. They had to provide a detailed description of each use case based on a template provided by the TAs.
6. *Class diagram*: Students were required to create a Class Diagram for their system.
7. *Implementation*: Students were required to implement all functionalities defined in the previous assignments, while ensuring each commit had a proper commit message and the README files were kept up to date. They were also instructed to keep issues up to date in their GitHub repository, using functionalities such as issue description, assignee, time estimates, milestones etc.
8. *Unit testing and debugging*: Students were required to perform unit testing for their backend functionalities and create a testing report with the test description and authors.
9. *Agile methodologies - Scrum simulation*: Students simulated Scrum development. Each group received a user story from the TAs, which they had to implement by the next class. They were required to break down the user story into tasks, estimate effort using planning poker and plan their sprint using an agile board on GitHub to help them plan and track progress. Finally, students were required to create a report containing development progress at the end of the week.
10. *Agile methodologies - Kanban Simulation*: In this assignment, students also received a user story from the TAs, which they had to implement, while following the Kanban methodology. During development the TAs asked for an update and sent the team new or updated requirements to simulate changes which the team had to adapt to.
11. *Continuous Integration/Continuous Deployment (CI/CD)*: Students were required to set up a CI/CD (Continuous integration/delivery/deployment) pipeline which would automate building the project and running unit test including setting up the database if necessary.
12. *Final Scrum simulation*: The final assignment required students to simulate multiple roles in scrum development. Each team was paired by another team in the role of the client and the developer. As clients, the team members provided a description of the feature, which the developer team had to implement. After implementation, a presentation was made for their clients who then gave feedback and rated the development team based on their progress.

3.2. Simulating the Development Lifecycle

The assignments were structured to simulate as many stages as possible of the SDLC while still adhering to the constraints of an educational setting. Although the SDLC typically starts with requirements analysis and system design before moving on to implementation and deployment, our approach deviates from this order, as the first two assignments include environment setup and coding. This was a deliberate trade-off to ensure that each student had sufficient technical skills before moving on to team assignments. The aim of this decision was to address the gaps in students' knowledge early and reduce the chance of team members not contributing due to lack of skills, which is a common problem, as individual differences in student teams are greater than in professional teams [5].

Initial group assignments focused on project **planning** and documentation, allowing teams to define their vision, scope, and goals. **Requirements analysis** followed later during the Scrum and Kanban simulations, where TAs took on the role of clients as suggested by [5], and introduced evolving user stories to simulate requirements changes and promote student autonomy. System **design** was addressed after initial coding began. Based on prior experience, we found students produced more accurate and coherent models when they had already implemented part of the system. Thus, use case and

Table 1
Assignment activities mapped to SDLC stages

SDLC stage	Activity	Relevant assignments
Planning & requirements	Glossary creation	3
	Application planning	4
	User story analysis and breakdown	9, 10, 12
Design	Use case diagram	4
	Use case descriptions	5
	Class diagram	6
Development	Environment setup	1
	CRUD operations	2
	Implementation of planned features	7
	Agile development and adaptation	9, 10, 12
Testing	Unit test implementation	8
	Testing report creation	8
Deployment	CI/CD pipeline setup	11
	Automation of build and test	11
Maintenance	—	—

class diagrams were introduced after some hands-on development. **Development** activities were integrated throughout the course, beginning with individual assignments and later continuing in team-based implementation phases. All group work was submitted via GitHub to support transparency and encourage the development of version control practices [7] and better simulate real-world development [12]. **Testing** was included in a dedicated assignment focused on unit tests and debugging. Due to time constraints and curriculum scope, we limited testing to backend functionality. While **deployment** and **maintenance** were not fully addressed, students did set up a CI/CD pipeline to automate building and testing, giving them exposure to automation tools commonly used in professional settings. Assignment activities mapped to SDLC stages are shown in Table 1.

4. Outcomes and Lessons learned

In addition to observations made by TAs during the course, we conducted an anonymous post-course survey among students (N = 22). The survey aimed to better understand students' perceptions of the course structure, team collaboration, and technical content. It consisted of a short set of Likert-scale items and open-ended questions. Students rated their satisfaction with different aspects of the course (e.g., the practical labs, group collaboration, and communication) on a 5-point agreement scale ranging from "Strongly disagree" to "Strongly agree." In addition, open-text prompts asked them to describe what they liked most, which tasks they found difficult, and how the course could be improved. This feedback complements our qualitative findings and provides additional insights into what worked well and where improvements are needed.

4.1. What worked well

The mix of **individual and collaborative assignments** proved to be a success, as very few team issues were reported. When asked, almost all students rated communication and work within their team as excellent or great and were happy with the amount of group work. In the survey, 8 out of 22 students (36%) selected "Strongly agree" and 14 (64%) selected "Agree" when asked whether they were satisfied with the practical part of the course. This means that all respondents expressed a positive evaluation, indicating a high overall level of satisfaction with the lab assignments (as shown in Figure 2).

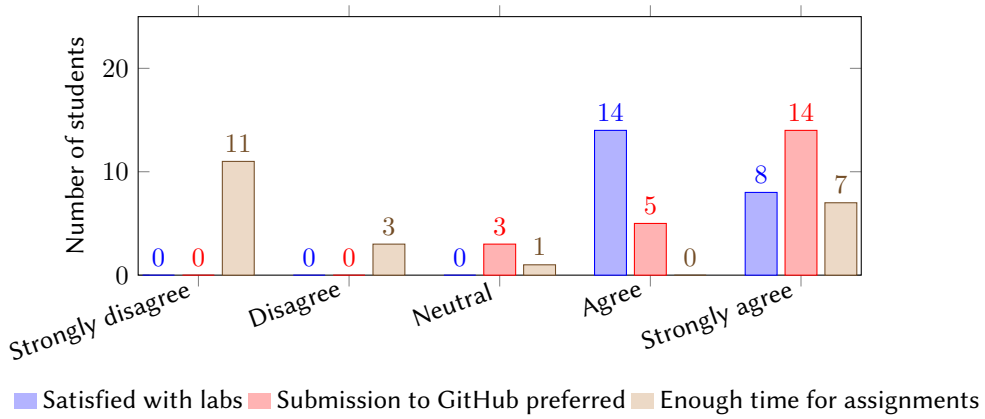

Figure 2: Student responses to selected post-course survey questions (N=22).

Allowing students to choose their teammates appears to have contributed positively to team dynamics and motivation.

The **gradual progression** from individual to group work helped students improve their technical skills before transitioning to collaborative development. This structure appeared to reduce common problems such as uneven contributions or technical dependency on stronger team members. Many students reported feeling better prepared for team projects due to the initial individual assignments.

Scrum and Kanban simulations were most frequently cited as the most valuable and enjoyable parts of the course, with 9 (41%) students naming them as their favorite assignments in free-text responses. They appreciated the structure they provide for task management and development. The students also felt that incremental assignment provided a better learning experience compared to isolated assignments, as they were able to gradually see their project improve.

The **use of GitHub** instead of traditional submission was another highlight. Students preferred assignments where no submission was required as everything was uploaded on GitHub. From the instructors' perspective, GitHub provided greater transparency into each team member's contributions, enabling fairer assessment and earlier detection of collaboration issues.

Overall, the course succeeded in simulating many of the practices and pressures of real-world development — from coordinating as a team and responding to changing requirements to managing technical complexity over time. While the simulation was not fully comprehensive, students were exposed to multiple interconnected stages of the SDLC, and many recognized the value of working within such a structure.

4.2. Challenges

While the course was well received, several recurring challenges were observed. They ranged from technical difficulties to team communication issues and workload management. These issues were also found in survey responses, where students most commonly mentioned limited project topic choices and tight deadlines. Many of these challenges are typical in collaborative student projects, but they still warrant targeted improvements in future course iterations.

Team communication proved problematic in certain groups, particularly when members were unavailable or using different messaging platforms. Some students began working late, making coordination harder. **Git** also posed issues early on, with students accidentally overwriting files or struggling with repository setup — especially those unfamiliar with version control systems.

Students also reported feeling rushed, particularly during individual assignments with **tight deadlines**. Additionally, the limited selection of **predefined project topics** left some students disengaged. On the instructor side, **tracking deadlines and contributions** via GitHub proved more labor-intensive than expected, as students tried to sneak in commits after the deadline, which meant that commits for each repository had to be manually reviewed.

Table 2
Identified challenges and planned improvements

Challenge	Planned improvements
Inconsistent team communication	Require a shared platform and communication plan (tool, channels, response times) [8]. Introduce mandatory weekly check-ins. [8].
Uneven Git proficiency	Start Git earlier with structured tutorials and a repository template.
Limited project choice	Define broader themes and a common set of required functionalities. Allow instructor-approved student proposals. Provide a small gallery of past exemplars to set scope/quality.
Deadline / contribution tracking	Use milestones/releases with fixed dates. Automate contribution analytics [9, 10, 13]. Provide a lightweight dashboard for TAs and teams [10].
Students don't see value in CI/CD	Introduce CI/CD earlier and tie it to ongoing releases. Require regular deployments alongside releases.
Documentation undervalued	Add a task requiring students to maintain or build on another team's code.
Peer feedback not honest	Make evaluations anonymous and structured to encourage honesty. Show the students sample projects to align ratings.
Unclear purpose of SDLC stages	Add a final reflection activity that helps students connect their work to specific SDLC phases [5]. Short quizzes/checkpoints on stage goals [5].

The CI/CD assignment, introduced late in the course, was seen as too difficult and disconnected from the rest of the workflow. Meanwhile, **documentation** was often treated as an afterthought, and **peer evaluation** lacked honesty due to fear of conflict. To address these issues, we have planned several adjustments, outlined in Table 2.

4.3. Lessons learned

We found that students engaged better when assignments formed a continuous line of work rather than isolated tasks. For instructors, this suggests that the order and coupling of tasks matters as much as the tasks themselves: short, linked iterations help students see progress and makes it easier for them to follow the course than when switching between unconnected assignments. This is consistent with reports that agile education benefits from short cycles and steady pace and that course structure needs to be aligned with iterative work [5, 6]. It also shows the importance of exposing students to software evolution and maintenance, rather than focusing only on building new projects from scratch, so that classroom work reflects how software actually changes over time [4].

Using GitHub made contributions and coordination clearer, and students felt this made teamwork fairer. At the same time, many struggled with the tool at first because the learning curve was higher than expected. For teachers, this shows two needs: give students an early introduction to collaboration tools, and use project data together with structured peer feedback to see how work is shared and to step in when needed. Earlier studies also show that GitHub can improve engagement and collaboration when used carefully [7, 12, 14], and that log data can reveal teamwork styles and uneven contributions that help teachers give feedback and make grading fairer [10, 13].

Students valued the agile simulations, but too much freedom at the start left some unsure about the process and what was expected. Teachers can make this easier by giving a bit of structure, such as examples of sprint planning, short checklists for roles and communication, and by bringing practices like CI/CD and documentation into the project earlier. This is in line with results from game-based Scrum simulations, which show that carefully designed structure helps students focus on core agile

practices without being overwhelmed [15]. Previous studies also point out that clear communication and visible processes are important in project courses [6, 8, 1].

A summary of key findings is listed below:

- **Students prefer using GitHub** for submissions but need training early on to master the platform.
- **The team communication plan** (communication platform, availability, check in time) should be defined and agreed upon at the start of group work.
- **Students prefer incremental work** on their project to isolated assignments.
- **Reviewing contributions on GitHub** is simple, but **time-consuming** if not automated.
- **Students value agile development** but need **guidance and specific instructions** to help them structure development as they can get overwhelmed by too much autonomy.
- **Peer evaluation** should be done **anonymously** and in a very **structured** manner to get honest feedback.
- Students tend to create very **superficial documentation** if they are not forced to use it, for example during peer handovers or team rotations.
- **CI/CD should be introduced early** and used throughout the course for the students to see the value in using it.
- **Predefined topics can limit creativity** and reduce engagement.
- **Individual assignments** are a great way to **improve the student's technical skills** before moving on to group work

5. Conclusion

Overall, the course successfully simulated many stages of the software development lifecycle and provided students with valuable experience in both technical and collaborative aspects of software development. The course structure—combining individual tasks with group projects—helped students gain essential skills for working in real-world development teams. Survey results and TA observations indicate that students especially appreciated the agile simulations and the use of GitHub, both of which made the workflow feel more realistic. At the same time, recurring difficulties with Git, tight deadlines, and the limited set of project topics show that even small design choices can strongly shape student motivation and learning. Documentation and peer evaluation also proved to be weaker points, often treated superficially unless they were directly tied to assessment.

These findings suggest that while the general approach of simulating the SDLC is effective, the way in which assignments are sequenced and connected plays an important role. Individual assignments early in the semester helped students build confidence and reduced dependency on stronger peers, while group tasks encouraged collaboration and communication. Agile methods were seen as valuable, but students also needed clear guidance to avoid becoming overwhelmed. The CI/CD assignment gave students useful exposure to modern workflows, but its placement at the very end meant that its value was not fully recognized.

By addressing these issues, future iterations of the course can provide an even more effective and engaging learning experience, better preparing students for the complexities of professional software development. In future work, we aim to explore how automated analytics of Git activity can support fairer and more efficient assessment of individual contributions in team-based projects. We would also like to examine how integrating CI/CD practices earlier and more consistently throughout the course affects students' understanding of modern development workflows. These findings are based on the first iteration of the redesigned course. In future iterations, we aim to collect and analyze data across multiple generations to strengthen the generalizability and empirical validity of the observed outcomes.

Declaration on Generative AI

During the preparation of this work, the authors used Grammarly for grammar, readability and spelling check. After using this tool, the authors reviewed and edited the content as needed and take full

responsibility for the publication's content.

Acknowledgments

The authors acknowledge the financial support from the Slovenian Research and Innovation Agency (Research Core Funding No. P2-0057).

References

[1] Y. Dubinsky, , O. Hazzan, A framework for teaching software development methods, Computer Science Education 15 (2005) 275–296. URL: https://doi.org/10.1080/08993400500298538. doi:10.1080/08993400500298538, publisher: Routledge _eprint: https://doi.org/10.1080/08993400500298538.

[2] What Is the Software Development Life Cycle (SDLC)? | IBM, 2024. URL: https://www.ibm.com/think/topics/sdlc.

[3] N. B. Ruparelia, Software development lifecycle models, ACM SIGSOFT Software Engineering Notes 35 (2010) 8–13. URL: https://dl.acm.org/doi/10.1145/1764810.1764814. doi:10.1145/1764810.1764814.

[4] J. Buchta, M. Petrenko, D. Poshyvanyk, V. Rajlich, Teaching Evolution of Open-Source Projects in Software Engineering Courses, in: 2006 22nd IEEE International Conference on Software Maintenance, 2006, pp. 136–144. URL: https://ieeexplore.ieee.org/document/4021331. doi:10.1109/ICSM.2006.66, iSSN: 1063-6773.

[5] V. Devedžić, S. R. Milenkovic, Teaching Agile Software Development: A Case Study, IEEE Transactions on Education 54 (2011) 273–278. URL: https://ieeexplore.ieee.org/document/5487434/. doi:10.1109/TE.2010.2052104.

[6] A. Schroeder, A. Klarl, P. Mayer, C. Kroiß, Teaching agile software development through lab courses, in: Proceedings of the 2012 IEEE Global Engineering Education Conference (EDUCON), 2012, pp. 1–10. URL: https://ieeexplore.ieee.org/document/6201194. doi:10.1109/EDUCON.2012.6201194, iSSN: 2165-9567.

[7] P. Patani, S. Tiwari, S. S. Rathore, The impact of GitHub on students' learning and engagement in a software engineering course, Computer Applications in Engineering Education 32 (2024) e22775. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/cae.22775. doi:10.1002/cae.22775, _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/cae.22775.

[8] C.-Y. Chen, P. P. Chong, Software engineering education: A study on conducting collaborative senior project development 84 (2011) 479–491. URL: https://www.sciencedirect.com/science/article/pii/S0164121210002931. doi:10.1016/j.jss.2010.10.042.

[9] K. Buffardi, Assessing Individual Contributions to Software Engineering Projects with Git Logs and User Stories, in: Proceedings of the 51st ACM Technical Symposium on Computer Science Education, SIGCSE '20, Association for Computing Machinery, New York, NY, USA, 2020, pp. 650–656. URL: https://doi.org/10.1145/3328778.3366948. doi:10.1145/3328778.3366948.

[10] N. Gitinabard, R. Okoilu, Y. Xu, S. Heckman, T. Barnes, C. Lynch, Student teamwork on programming projects: What can GitHub logs show us?, Proceedings of the ACM Technical Symposium on Computer Science Education (SIGCSE) (2020) 1007–1013. URL: https://doi.org/10.1145/3328778.3366897. doi:10.1145/3328778.3366897.

[11] Build software better, together, 2025. URL: https://github.com.

[12] C. Hsing, V. Gennarelli, Using GitHub in the Classroom Predicts Student Learning Outcomes and Classroom Experiences: Findings from a Survey of Students and Teachers, in: Proceedings of the 50th ACM Technical Symposium on Computer Science Education, SIGCSE '19, Association for Computing Machinery, New York, NY, USA, 2019, pp. 672–678. URL: https://dl.acm.org/doi/10.1145/3287324.3287460. doi:10.1145/3287324.3287460.

[13] S. Hamer, C. Quesada-López, A. Martínez, M. Jenkins, Measuring students' contributions in software development projects using Git metrics, in: 2020 XLVI Latin American Computing

Conference (CLEI), 2020, pp. 531–540. URL: https://ieeexplore.ieee.org/document/9458363. doi:10.1109/CLEI52000.2020.00068.

[14] M. A. Nelson, L. Ponciano, Experiences and insights from using github classroom to support project-based courses, in: 2021 Third International Workshop on Software Engineering Education for the Next Generation (SEENG), ????, pp. 31–35. URL: https://ieeexplore.ieee.org/document/9474647. doi:10.1109/SEENG53126.2021.00013.

[15] C. G. von Wangenheim, R. Savi, A. F. Borgatto, SCRUMIA—an educational game for teaching SCRUM in computing courses 86 (????) 2675–2687. URL: https://www.sciencedirect.com/science/article/pii/S0164121213001295. doi:10.1016/j.jss.2013.05.030.