POLITECNICO DI TORINO
Repository ISTITUZIONALE

Teaching Scrum with a focus on compliance assessment

Original
Teaching Scrum with a focus on compliance assessment / Torchiano, Marco; Vetro, Antonio; Coppola, Riccardo. - STAMPA. - (2024), pp. 595-599. (Intervento presentato al convegno 28th International Conference on Evaluation and Assessment in Software Engineering (EASE 2024) tenutosi a Salerno (ITA) nel June 18--21, 2024) [10.1145/3661167.3661277].

Availability:
This version is available at: 11583/2987999 since: 2024-06-27T15:24:13Z

Publisher:
ACM

Published
DOI:10.1145/3661167.3661277

Terms of use:

This article is made available under terms and conditions as specified in the corresponding bibliographic description in the repository

Publisher copyright

(Article begins on next page)

12 October 2024

Teaching Scrum with a focus on compliance assessment

Marco Torchiano
marco.torchiano@polito.it
Politecnico di Torino
Torino, Italy

Antonio Vetrò
antonio.vetro@polito.it
Politecnico di Torino
Torino, Italy

Riccardo Coppola
riccardo.coppola@polito.it
Politecnico di Torino
Torino, Italy

ABSTRACT

The Scrum framework has gained widespread adoption in the industry for its emphasis on collaboration and continuous improvement. However, it has not reached a similar relevance in Software Engineering (SE) curricula. This work reports the experience of five editions of a SE course within an M.Sc. Degree in Computer Engineering. The course primary educational objective is to provide students with the skills to manage software development projects with Scrum. The course is based on the execution of a team project and on the definition of qualitative and quantitative means of assessment of the application of Scrum. The conduction of five editions of the course allowed us to identify several lessons learned about time budgeting and team compositions in agile student projects and its evidence of the applicability of the framework to software development courses.

CCS CONCEPTS

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

KEYWORDS

Software Engineering Education, Agile Methods, Scrum, Measurement

ACM Reference Format:
Marco Torchiano, Antonio Vetrò, and Riccardo Coppola. 2024. Teaching Scrum with a focus on compliance assessment. In *28th International Conference on Evaluation and Assessment in Software Engineering (EASE 2024), June 18–21, 2024, Salerno, Italy.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3661167.3661277

1 INTRODUCTION

Scrum is an agile framework primarily used in software development management, characterized by its iterative approach and emphasis on collaboration, adaptability, and continuous improvement. In current practice, Scrum has gained widespread recognition for its ability to enhance productivity, efficiency, and transparency in various tasks, including – but not limited to – software development. Its importance in university education lies in its capacity to cultivate essential skills such as teamwork, communication, problem-solving and critical thinking, preparing students for the dynamic and rapidly evolving demands of the modern workforce.

Recent evidence from scientific literature underlines that Scrum is not always considered a cornerstone for modern software development by Software Engineering curricula and that in many cases reference curricula do not consider Scrum with a great relevance in knowledge areas nor in the time scheduled for it [5].

However, several pieces of research in Software Engineering education provided successful examples of integrating the teaching of Scrum fundamentals in development courses. Experience reports about the application of Scrum in developing a project in an undergraduate Software Engineering capstone course underline a positive appreciation by the involved students [3], and the possibility of obtaining student projects that can meet real-world quality standards and be adopted for use by Universities after the termination of the course [1]. The construction of a development setup using Scrum also allows courses to involve IT professionals, allowing students to gain a better appreciation of the inherent challenges involved in crafting larger and more realistic software applications than traditional, small-scale course projects [2].

This manuscript reports the experience of introducing the teaching of the Scrum framework in a Master's level Software Engineering course and describes an assessment approach, both qualitative and quantitative, to evaluate the outcomes of the students' development activities as well as the coordination and collaboration within teams. We detail the organization of the course and the results of its first five editions. On this basis, we discuss the educational results obtained and the key lessons learned.

2 COURSE ORGANIZATION

In this section we describe the way the course is organized, emphasizing the educational goals, the outline of the course, the way the teams were composed, and the project(s) assigned to the students.

2.1 Educational goals

The primary educational objective of the course is to provide the students with the essential skills and knowledge required to effectively manage and execute software projects using agile methodologies. Our focus is on instilling a comprehensive understanding of the agile development approach, emphasizing hands-on experience in carrying out team-based software projects efficiently and in an inclusive manner, i.e. enhancing the peculiarities of each person.

In pursuit of this educational goal, we adopt the Scrum framework as a reference agile methodology: Scrum, in contrast to other agile methodologies like Kanban, provides a well-structured and systematic framework that is particularly suitable for those who are new to agile practices. Scrum's emphasis on iterative development facilitates a smoother learning curve for participants, allowing them to grasp the fundamental concepts of agile project management in

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).
EASE 2024, June 18–21, 2024, Salerno, Italy
© 2024 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-1701-7/24/06
https://doi.org/10.1145/3661167.3661277

a more structured manner. Scums's regular checkpoints are helpful to receive continuous feedback and adapt to teachers' indications, allowing students to conduct a reflective practice led by the critical analysis of their work. In addition, the focus on collaborative teamwork reduces students' dropouts and increases the overall achievement of learning outcomes.

Although the main object of the course is the process and not the technology, we devote the majority of time to hands-on experience for delivering working software: this is crucial for developing the skills necessary to handle the complexity of software project management in times of dynamic environments and volatile requirements. In summary, at the end of the course, the students acquire basic knowledge of: (i) the principles of agile development and how they are implemented in Scrum; (ii) how to manage a team-based software project, planning activities and adapting to changes during its evolution; (iii) how to critically analyse the processes involved in software development and improve them; (iv) how to set up self-organizing teams and arrange teamwork combining efficiency and inclusiveness; (v) how to manage stakeholders' collaboration and report on project's achievements and fulfilment of requirements; (vi) how to manage and improve software quality.

2.2 Course outline

The course lasts 14 weeks and it is conceptually divided into two phases: a *how-to-do* phase (6 weeks) and a *do* phase (8 weeks). Each week includes 4,5 hours of lecture/activity, three of which are held in parallel rooms: in total, each student attends 60 hours of lecture, while the overall teaching effort is about 150 hours.

In the *how-to-do* phase, we introduce the theoretical principles of Agile and we show how a typical Scrum development process is organized. We carry on a pilot project through a series of workshops focused on the key activities in the Scrum method, each corresponding to a workshop assigned to the students: W1 - User stories, W2 - Estimation, W3 - Planning, W4 - Daily scrum, W5 - Review, W6 - Retrospective.

Each workshop is introduced by a review of the corresponding Scrum activity and related artefacts (e.g., the Scrum board) and instructions on how to perform the activity. Then each team performs the activity in class: during the activity, they receive feedback and guidance from the three instructors. At the end, teams gather again in a plenary session where instructors summarize the most frequent errors and difficulties encountered, and how to face them.

In the *do* phase the students carry on the real project (see Sec. 2.4). In this phase, in addition to the project's related activities, additional theoretical lectures are given on a variety of ancillary topics: agile contracts, technical debt (TD) and software quality measurement, other agile approaches, and free software licenses. Practical activities are also conducted in conjunction with some of these additional lectures, for example on managing TD, choosing a proper free software license or making an Elephant Carpaccio exercise.

2.3 Team composition

We form teams right after the first week. As inclusiveness is a fundamental goal in our course, we strive to achieve homogeneity in teams by balancing a variety of aspects that are assessed through an online form: provenance and nationality; gender; working student; knowledge of basic programming languages (Java, Python, SQL, Cpp), knowledge of web development languages and techniques (e.g., HTML, JS, Restful APIs, Docker, etc.); previous knowledge on software engineering concepts and tools (e.g., UML, testing, code reviews, GIT), previous knowledge on agile. The goal is to avoid the isolation of the less technically skilled, non Italian students, etc.

Through the years, the total number of teams ranged from a minimum of 11 in the first edition (A.Y. 2019/20, 62 students) to a maximum of 20 in the A.Y. 2022/23 (122 students). In the last edition (i.e., A.Y. 2023/24) there were 119 students organized in 18 teams. Attendance to workshops and to the project's reporting appointments (see Sec. 2.4) is mandatory, we allow only four absences.

2.4 The project assignment

During the *do* phase of the course, the students carry on four two-week sprints on the real project. In the different editions of the course, the students have been tasked with the full-stack development of web applications, with varying domains and requirements throughout the years, e.g. a thesis management system for a university, a hike tracker application, or a solidarity purchasing group. During the project, one of the teachers acts as the Product Owner and answers the questions on a Telegram chat so that every team is informed of the feedback. Another teacher plays the role of Scrum Master and is available to teams for counselling on how to apply Scrum or to discuss and help resolve collaboration issues. Each person must devote 16 hours of work during a sprint.

At the end of every sprint, teams carry on a product review and a sprint retrospective. In product reviews, teams present in front of one instructor and another team (both randomly assigned at each sprint) which plays the role of stakeholders: after the 20-minute demo of the working software, they provide feedback as a pair of positive aspects (e.g., high usability, a well working functionality, etc.) and a negative aspect (e.g., a bug noticed during the demo, or a user story not well developed, etc.), the latter of which must be reported in the GitHub repository and properly addressed in the next sprint. The teams then swap: stakeholders become presenters and vice-versa. During both reviews, the instructor makes questions to assess the correct development of user stories and other aspects and takes notes to be subsequently analyzed (see Sec. 3). Notice that the definition of done adopted in the course includes the following aspects: unit tests passed; code reviewed performed; code on the version control system; E2E tests performed. Teams might add additional elements at their will.

Concerning the sprint retrospectives, teams have 20 minutes to summarize the content of the *retrospective report* and to answer questions from the teacher (randomly assigned at every sprint). The retrospective report is a template[1] to be filled with a mix of quantitative and qualitative data, on the following aspects: stories developed, effort and overall estimation error; tests performed and related effort; technical debt effort and achievements (sprint 3 and 4 only); reflections on estimation errors and their causes, coordination issues, improvements achieved and planned for the next iteration.

Students can use technologies of their choice, both for development and for supporting team coordination (e.g., many use Discord

[1] It can be accessed at https://doi.org/10.5281/zenodo.11032216

channels to make Scrum meetings). However, they are required to include the following technologies/tools in their development process: YouTrack for tracking tasks, effort and project progress; GitHub as hosting repository; SonarCloud to manage TD; Docker to containerize the application in two releases (at sprint 2 and at sprint 4). All repositories (GitHub, SonarCloud, YouTrack, Docker Hub) need to be public for assessment purposes and for encouraging collaboration even among teams. At the end of the project, teams must upload a 7-minute teaser video on YouTube which shows how the application performs a provided scenario. The students can use part of the time also to show the strengths of their application.

3 ASSESSMENT METHOD

The grading is performed by assessing both team and individual performance. Team assessment weights 80%, while individual assessment weights 20%. The individual assessment entails both theoretical knowledge and contribution provided to teamwork, especially during the demos and retrospectives: notes taken by instructors during sprint reviews and retrospectives contribute to this part of the evaluation.

The team assessment is performed using the following criteria:

- *Team coordination and improvement*, which includes the proper use of the project tracking tool (YouTrack), planning and time management, and improvement of corresponding tracked metrics (see Sec. 3.1) during the project;
- *Quality* of sprint reviews (demos) and retrospectives meetings;
- *Doneness*, which means testing of product, management of technical debt, and deployed releases.

3.1 Assessment of Teams

Teams are evaluated using a combination of qualitative and quantitative techniques. The quantitative assessment is performed through a team assessment card, an example is reported in figure 1). Qualitative assessment is carried on on the basis of notes taken during project reviews and sprint retrospectives.

3.1.1 Coordination and improvement. The criteria we used to assess the team coordination are:

- Compliance with the assigned time budget. Looking at the "Team sprint effort" (top right of the card) we check if the effort is close to the time budget 16 ± 1 hour per sprint per person. We focus on the latest three sprints and the overall trend.
  In the sample card, we observe that the effort is within the acceptable range and getting closer to the target.
- Effort balanced among members. We used the "Individual sprint effort" heat map (second from the top on the right) to detect anomalies. Moreover, we compute the overall imbalance using the Gini index, where good values are below 3% and bad values are above 9%.
  In the sample card, we appreciate visually a good balance that is confirmed by the Gini index of 2.2%.
- Uniform distribution of effort during the sprint. The "Team daily effort" line chart (top left of the card) shows the trend of effort, including a per sprint Pearson regression coefficient. We can spot strong upward behaviours visually, moreover, we consider r values larger than 0.3 as suspect, and above 0.5 as critical.
  In the sample card, we can see a healthy behaviour with r values below the warning thresholds.
- Small task size. The "Estimated normalized task size" and "Actual normalized task size" boxplots (third row in the card) report the distribution of task size normalized per number of participants. The diagrams feature a reference line corresponding 2 hours per participant which we recommended as the upper limit during the course. In general, we consider a good result when teams manage to keep three-quarters of the task below the two-hour threshold. We mainly look at estimation but we also check whether the actual values are kept under control.
  In the sample card, we observe a very good size of tasks.
- Accurate estimation error. The "Mean absolute relative estimation error" (MRAEE) diagram (second from bottom on the left column) reports the error in the four sprints and the overall average with the orange line. In general, we deem excellent MRAEE below 20% and consider critical an error above 50%, especially in the latest two sprints.
  In the sample card we observe an initial large estimation error that has been reduced during the project. With an overall average below 30%.
- Complete estimation of tasks. The "Active tasks without estimation" (second from bottom on the right column) reports the percentage of active tasks – i.e. tasks with reported effort – that have no estimation. We usually tolerate a very small number of non-estimated tasks, while in general, we consider critical more than 5% of non-estimated tasks.
  In the sample card, we observe the ideal condition of all tasks being estimated.
- Tracked planning and scrum meetings. The diagrams "Tracked meetings" and "Tracked planning meetings" (bottom two diagrams in the card) show the average time of scrum and planning meetings respectively. The diagrams also report the number of people involved and the number of occurrences. We check, first of all, that meetings are effectively tracked, and also that possibly all members of the team participate. We consider it acceptable that one member is missing in one meeting considering possible personal issues during the semester.
  In the sample card, we observe scrum meetings getting shorter and more frequent – which is a positive fact – and one person missing from the latest planning meeting – which is still acceptable –.

We assessed improvement considering the trend along the four sprints of the above criteria.

3.1.2 Quality of meetings. We took personal notes during the product review and sprint retrospective meetings that were used to provide immediate feedback to the teams and later to assess the ability of the teams.

- Concerning the product reviews, we evaluated the capability to clearly explain which user stories were implemented, to

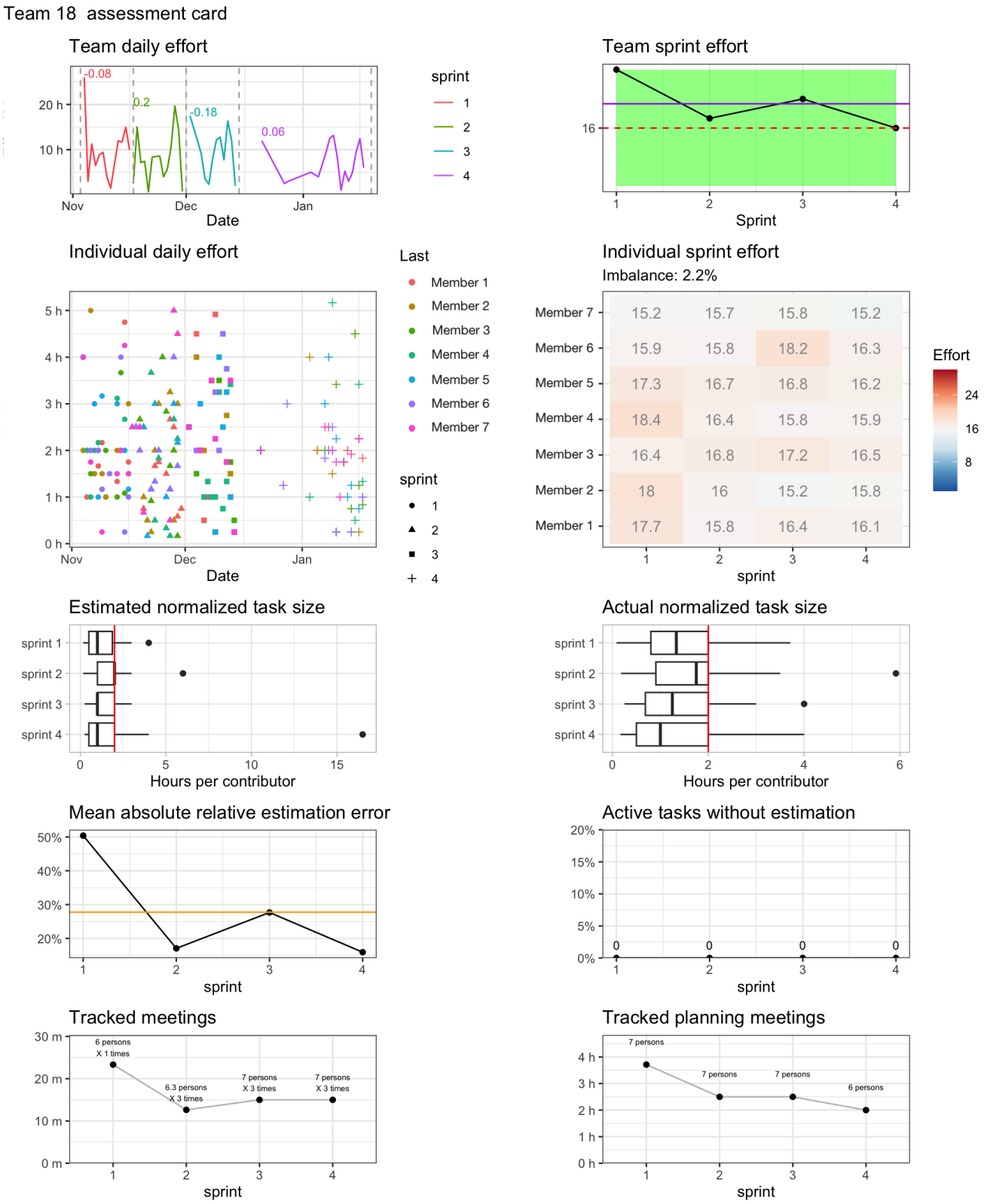

Figure 1: Sample of Team assessment card.

demonstrate them properly on the application, the promptness in the responses to comments from stakeholders, the clarity in the description of the product, and the ability in self-evaluation.
- Concerning the retrospective, we evaluated the ability to analyze the team operation during the past sprint, recognize the problems and mistakes committed, and eventually propose sensible improvement actions.

3.1.3 Doneness. Based on our Definition of Done, we assessed the level of *doneness* of the products delivered at the end of the four sprints, checking whether: the teams performed enough unit and E2E testing and evidence is found in the GitHub repository (e.g., source code of tests, test plans, coverage reports); no software failures appeared during sprint reviews (with a tolerance of one occurrence); TD was managed, i.e. there are explicit tasks on TD reduction that are consistent with the SonarCloud web reports; a running docker image was available in sprints 2 and 4.

3.2 Assessment of Team Members

We evaluated the individual contribution using both quantitative information – based on the team assessment card – and qualitative information based on the notes taken during the meetings and the evidence in the repositories. The main criteria we used are:

- Active participation in meetings: we assessed this aspect using our notes about how often the student spoke during the meetings, we asked each member to present at least once;
- Effective involvement in teamwork: based on the participation we assessed whether the student appeared to be knowledgeable of the overall teamwork, and double-checked it with a few questions on performed activities;
- No outliers in daily personal effort: looking at the "Individual daily effort" diagram in the assessment card we checked that the student did not report any abnormal daily effort (e.g. 14 hours in a single day);
- Sprint effort within acceptable range: based on the "Individual sprint effort" diagram we checked whether the student reported an effort too far from the individual sprint budget (i.e. 16 hours over two weeks).

4 LESSONS LEARNED

The five editions of the course that were run allowed us to sum up some lessons learned about the students' work organization and the way to track and assess it.

A typical struggle for students is complying with the time budget. A relevant amount of errors in coordination are related to students' time tracking. A typical correction that is applied by students is the *bulk* addition of allocation to tasks at the end of a sprint, in place of rigorous time tracking for all activities performed during the week.

By analyzing the improvement in coordination, we acknowledge that most of the teams learn how to self-organize the work by the second sprint. This aspect can be seen as evidence of the feasibility of successfully applying the Scrum framework on a course project, but with an inevitable learning time that includes the duration of the pilot project with workshops (about 2 weeks) and the first sprint (2 weeks) of the project work, so overall one month.

We observe that teams tend to self-organize themselves in two sub-teams, dedicated respectively to the back-end and the front-end. This division proves to be not successful since if any of the sub-team is slower in performing tasks, entire sets of stories are not done by the team.

Although the effort is fixed, a few teams suffer from self-generated pressure of high productivity, which results in many stories developed without conforming to the definition of done. Usually after sprint 2 they are able to better manage this aspect.

The mandatory time tracking activity made the effort balanced between members in most groups, with very few slackers. This aspect is positive in avoiding hostilities among team members.

In agreement with the DevX framework [4] we observed that teams that managed to keep task size small – thus shortening feedback cycles – were able to achieve better results.

5 CONCLUSIONS AND ACTIONABLE TIPS

The conduction of five editions of the course allowed us to positively assess the feasibility of teaching Scrum to Master's students with effectiveness in learning and positive teamwork and cooperation. We identify a few main actionable tips for educators aiming at setting up Scrum projects for their students:

- Do not set a target on scope but only on time (and cost). This aspect proved to be beneficial to avoid teams rushing to complete all the requirements thus paying not enough attention to quality-related activities;
- Provide a specific time frame for the course project. This aspect proved to be beneficial to better engage and motivate the students, and avoid situations where entire teams were unable to pass the course because of less-producing members.
- Balance teams according to a variety of factors (skill set, country of origin, gender) to maximize diversity within groups. This aspect usually pays back and prevents dropouts or very low individual performances.
- In case of team coordination problems, the Scrum master should intervene as soon as possible, to avoid recrudescence of problems and unmanageable/unrecoverable situations.

For future editions of the course, our plan is to introduce acceptance criteria specific to the main user stories to have a finer grain control on the delivered projects. We also plan to add universal and technology-agnostic automated End-to-end test cases to verify them. Finally, we will aim at hosting a yearly industrial seminar on how Scrum is applied in a company in the daily practices.

REFERENCES

[1] Dabin Ding, Mahmoud Yousef, and Xiaodong Yue. 2017. A case study for teaching students agile and scrum in Capstone course. *Journal of Computing Sciences in Colleges* 32, 5 (2017), 95–101.
[2] Panagiotis K Linos, Ryan Rybarczyk, and Nathan Partenheimer. 2020. Involving IT professionals in Scrum student teams: An empirical study on the impact of students' learning. In *2020 IEEE Frontiers in Education Conference (FIE)*. IEEE, 1–9.
[3] Viljan Mahnic. 2011. A capstone course on agile software development using scrum. *IEEE Transactions on education* 55, 1 (2011), 99–106.
[4] Abi Noda, Margaret-Anne Storey, Nicole Forsgren, and Michaela Greiler. 2023. DevEX: What Actually Drives Productivity? *Commun. ACM* 66, 11 (2023), 44–49. https://doi.org/10.1145/3610285
[5] Ricardo Pérez-Castillo, Ismael Caballero, and Moisés Rodríguez. 2018. Improving the experience of teaching Scrum. In *2018 IEEE Global Engineering Education Conference (EDUCON)*. IEEE, 1598–1605.