Scrum2Kanban: Integrating Kanban and Scrum in a University Software Engineering Capstone Course

Christoph Matthies
Hasso Plattner Institute, University of Potsdam
Potsdam, Germany
christoph.matthies@hpi.de

ABSTRACT

Using university capstone courses to teach agile software development methodologies has become commonplace, as agile methods have gained support in professional software development. This usually means students are introduced to and work with the currently most popular agile methodology: Scrum. However, as the agile methods employed in the industry change and are adapted to different contexts, university courses must follow suit. A prime example of this is the Kanban method, which has recently gathered attention in the industry. In this paper, we describe a capstone course design, which adds the hands-on learning of the lean principles advocated by Kanban into a capstone project run with Scrum. This both ensures that students are aware of recent process frameworks and ideas as well as gain a more thorough overview of how agile methods can be employed in practice. We describe the details of the course and analyze the participating students' perceptions as well as our observations. We analyze the development artifacts, created by students during the course in respect to the two different development methodologies. We further present a summary of the lessons learned as well as recommendations for future similar courses. The survey conducted at the end of the course revealed an overwhelmingly positive attitude of students towards the integration of Kanban into the course.

CCS CONCEPTS

• **Applied computing** → **Collaborative learning**; • **Software and its engineering** → **Agile software development**;

KEYWORDS

Agile methods, capstone course, Scrum, Kanban

1 INTRODUCTION

Teaching software development methods and processes in capstone courses using hands-on projects is a widely adopted practice in universities [14]. These projects strengthen comprehension and allow students to apply their theoretical process knowledge in real-world software projects [8, 20, 21]. The 2013 CS curriculum guidelines jointly published by ACM and IEEE state that the opportunity for students to iteratively work through a development cycle, assessing outcomes and applying their gained knowledge positively impacts learning success [29]. Teaching iterative agile and lean software development methods has thus, in addition to fulfilling industry needs, become an important issue when designing software engineering university courses [13]. Most of these courses focus on the Scrum methodology [21, 22, 25], as it is currently the most popular process employed in industry settings [31] and has become mainstream [9]. Much of the innovation in the domain of software development processes happens in industry, driven by a need for practices that increase software quality and enable development teams to work together more effectively. Therefore, the software engineering curricula must respond to changes in how agile and lean software development methods evolve in practice.

While in most organizations Scrum is still the most employed development methodology, lean approaches such as Kanban are on the rise. The latest "State of Agile Report" by VersionOne, published in 2017, identified Scrum and Scrum-XP-Hybrids as the most common agile methodologies, used by 68% of respondents. These methods were followed directly by custom hybrids (8%), Scrumban (8%) and Kanban (5%) [31]. Other recent surveys confirm this trend. Komus et al. state that in their survey of more than 1000 respondents, Scrum is the most often employed agile methodology (85% of respondents), followed by Kanban and Lean [7]. The Scrum Alliance reports that 89% of respondents to their "State of Scrum" survey named specifically Scrum as the agile approach or at least one of the agile approaches used in their organization [28]. Again, Kanban was the second most commonly named software development approach.

In order for university software engineering courses to stay relevant and up to date, Kanban and lean software development approaches need to be integrated into the curriculum. To this end, we present an undergraduate project course design, based on a modified version of Scrum, which introduces Kanban towards the end of the instruction period and allows students to gain a better overview of agile methods, in line with recent research [13]. We evaluate our approach both with surveys as well as analysis of the produced development artifacts, e.g. commits and user stories/tickets.

The following research questions guide our work:

RQ1 How can the Kanban software development methodology be integrated into an agile hands-on undergraduate university software engineering course?

RQ2 What are students' perceptions of Kanban practices and the proposed teaching approach?

RQ3 Are students' perceptions reflected in the development artifacts that were produced while employing Kanban?

RQ4 What influence does using Kanban have on the employed workflow during the course?

The rest of the paper is structured as follows: Section 2 briefly introduces the agile methods Scrum and Kanban as well as possible hybrids employed in the industry. Section 3 describes the course design and details the development process that is followed during the course. The following section, Section 4, describes how we evaluated the chosen teaching approach using surveys and analysis of development data. Section 5 details related work in the research areas of teaching agile capstone courses and evaluating development processes. The last section, Section 6 concludes and summarizes our findings.

2 SCRUM, KANBAN AND HYBRIDS

Scrum as well as Kanban are agile software development methods. They focus on collaborative teamwork and explicitly acknowledge the importance of self-organization and empowerment of development teams [6]. Scrum is the more well-known of the two approaches and relies on an iterative and incremental development cycle, with the goal of producing an increment of potentially shippable software at the end of each "Sprint" iteration. It focuses on managing projects by introducing defined roles (e.g. Product Owner), meetings (e.g. Sprint Retrospective) as well as organizational and process artifacts [27]. The Kanban development method is inspired by the Toyota Production System [18] and the principles of Lean manufacturing [33]. In contrast to Scrum, it is less descriptive and focuses on visualizing workflows using Kanban boards, limiting the work in progress and ensuring that work flow as fast as possible through the system by removing bottlenecks [6].

Due to their different focuses, Scrum practices and Kanban principles can be combined, resulting in a development process often referred to as *Scrumban* [11]. In a study on the adoption of Scrum in enterprises Kapitsaki et al. identified the need for more studies that follow Scrum's adoption and its emerging variants, especially in combination with Kanban [5]. In a 2015 literature review on the topic of process models in practice Theocharis et al. state that hybrid approaches, including "traditional-agile" as well as "agile-agile" hybrids represent a common model of use [30]. They further state that in their set of 22 selected papers on process usage, only five provided quantitative data and call for more instruments for process tailoring and combination. A recent online study, performed in 2018 by Kuhrmann et al. on the characteristics of hybrid development approaches confirmed these findings. The authors state that among the 69 surveyed practitioners across Europe, hybrid development methods were widely used, regardless of company size or industry domain and were applied even in when company-wide policies for process use were present [10]. As such, teaching Kanban and its possibilities and combinations is vital for educating the next generation of well-rounded software engineers.

#	Learning target
1	Experience with Kanban as well as Scrum and all of its artifacts and meetings
2	Knowledge of how to scale Scrum over multiple collaborating teams
3	Ability to use BDD and TDD where appropriate
4	Confidence with branching and merging in a source code management (SCM) system
5	Experienced the value of continuous integration (CI)
6	Learned to critically self-assess one's own role in a team

Table 1: Overview of the main learning targets of the software engineering II course.

3 COURSE DESIGN

In order to answer our first research question (*RQ1*) on how Kanban can be integrated in a university software engineering course, we describe an undergraduate capstone course design, which was developed over the last 4 years and was most recently taught in the winter semester of 2017/18. Its main focus is collaboration and self-organization in teams and in a team-of-teams using agile methods as well as modern software development best practices. The course emphasizes hands-on learning in a simulated real-world scenario, its main learning targets are summarized in Table 1. All students of the course, split into teams, jointly develop a single software system under an open-source license, hosted on the collaboration platform GitHub[1]. Work in the project is accompanied by regular lectures as well as support by tutors (junior research assistants), who are present during students' team meetings. All materials and slides of the latest iteration of the course are available online[2]. As the course is recommended for students in the last semester of undergraduate studies, students have already attended lectures on the fundamentals of software engineering. The Scrum method is employed in four sprint iterations in the beginning of the course, as its more descriptive nature and requirements for structure lend themselves to introducing agile concepts [13]. After students have gained experience with the employed technologies and have become familiar with their teams, the more dynamic Kanban method is employed. This shift in process introduces additional challenges and must be accompanied with tutoring of student teams as well extensive lectures.

3.1 Course Schedule

The course described here is typically run in the winter semester with a length of 15 weeks. It features both traditional lectures, introducing students to agile methods and best practices, as well as stretches of project work and separate exercises, Table 2 lists a typical schedule. Lectures (e.g. on Scrum and git) are more frequent towards the beginning of the course in order to prepare students for the hands-on project. Later lectures focus on tips and tricks, guest lectures from industry practitioners and introduce the Kanban method. During the project part of the course, students jointly

---

[1]https://github.com/
[2]https://hpi.de/plattner/teaching/winter-term-201718/softwaretechnik-ii.html

Week	Project work	Lectures
1		• Course introduction • Technology tutorial
2-4	Introduction exercise	• Scrum methodology • LEGO exercise • Software testing • Git version control system
5-6	Sprint 1	
7-8	Sprint 2	• Code Reviews • Deployment
9-11	Sprint 3 & Interm. presentation	• Scrum Tips • Guest Lecture
12-13	Sprint 4	• Guest lecture • Lean Software and Kanban
14	Kanban Week	
15	Final Presentation	• Summary, exam preparation

Table 2: Course schedule. Project work is performed by students in their teams or individually on their own schedule.

develop a web-application using the Ruby on Rails framework[3] in four Scrum sprints, followed by a week in which the Kanban method is practiced.

In order to allow students to get started with hands-on project work as fast as possible, two practical exercises are employed at the beginning of the course: A LEGO Scrum exercise and an automated programming tutorial.

After a lecture introducing the basics of Scrum, the roles, meetings and artifacts involved, a 90 minute "fast-track" Scrum simulation exercise is set. This sort of exercise, where LEGO is built in sprints of a few minutes each, has been shown to engage students, simulating the phases and difficulties of real software projects and advances student learning [19].

Furthermore, an interactive programming exercise aims to familiarize students with the web application framework used in the following development project. The exercise, which students have 3 weeks to complete, involves an automated Continuous Integration (CI) server, which analyzes students' progress and assigns them new tasks through GitHub [16]. Students are thus introduced not only to the programming framework, but also all the other tools necessary, such as dependency managers, testing frameworks, editors and CI servers.

3.2 Scrum Development Process

During the four sprints of the project, a version of the Scrum process, adapted to the time constraints of students is employed, see Figure 1. Participants form self-teams of 5 to 8 members, each with their own Product Owner (PO) and a Scrum Master (SM), roles which are also performed by students. All other students are developers. During each week of the course, students are expected to spend 8 hours of work time, including lectures and team meetings. Due to these time constraints, daily Scrum stand-up meetings are replaced by weekly versions.

---

[3]http://rubyonrails.org/
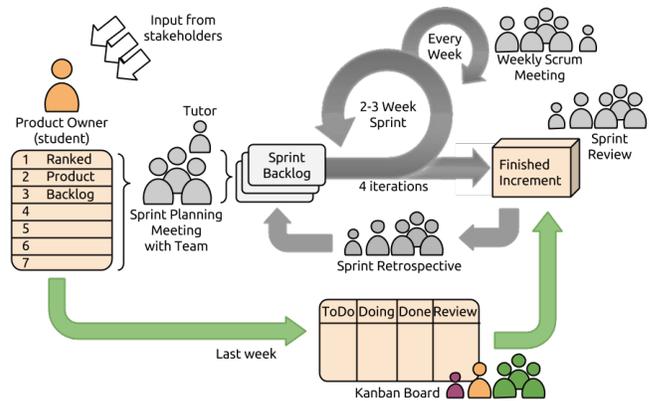
Figure 1: Overview of the modified Scrum process (top, grey) and the integration of Kanban (bottom, colored) used in our software engineering course.

For every sprint, a Scrum planning, sprint review, retrospective as well as a weekly stand-up meeting is organized by the students. These meetings are held in the presence of a tutor, who is able to answer questions and provide guidance and advice, if needed. In order to elicit the initial requirements for the system to be developed during the project, the POs meet with a member of the teaching staff, the customer. Guided by an instructor, taking the role of "chief PO", the product backlog is filled with user stories. The Product Owners then present the product vision to their teams and the first Sprint is started. During the middle of the project, the POs hold an intermediate presentation, detailing the current status of the software to the entire course as well as external stakeholders that will use the software.

3.3 Kanban Development Process

Kanban is considered, relatively speaking, less *prescriptive* and more *adaptive* than Scrum, meaning there are fewer rules to follow [6]. This means Kanban introduction and teaching is better suited when students have already acquired an understanding of agile concepts and can interpret and adapt the process according to their needs and context. Therefore, in order to improve the learning of Kanban, it is advisable to introduce the method after the students have already gained experience with Scrum during the project [13]. In fact, Kanban users are expected to experiment with and continually improve their process in order to customize it to their environment, a task that is beyond simple application of learned methods and practices [13, 17]. While it can be argued that systems should be stable, i.e. a development process in a team that has a long history, before a new process is introduced, Kanban can also provide mechanisms to achieve the needed stabilization by focusing on throughput [24].

Towards the end of the course, after students have become familiar with Scrum and their teams, i.e. the *norming* or *performing* stages of group development [4], the ideas of Lean Software development and Kanban are introduced in a lecture, see Table 2. Kanban boards and the concepts of visualizing workflow and limiting work in progress (WIP) through WIP limits are presented in detail. students

discuss the advantages and drawbacks of the method, compared to Scrum and other processes they previously experienced. The course attendees are then encouraged to try out these new ideas in their project by employing Kanban for the last week to 10 days of the project, instead of a last Scrum sprint, see Figure 1. Kanban introduction is furthermore most suited towards the end of the course, due to the natural way of how work is performed near the deadline of the project. As the course end and the date of the final presentation are known in advance, students' focus in the last few project weeks traditionally shifts from a "feature implementation" to a "bug fixing and polishing" mode, in order to deliver a usable project that can be presented. This work mode entails dynamic and often changing new requirements handed to the developers by the Product Owners, as bugs are fixed, new ones are uncovered, and the software is made ready for presentation. Therefore, the concept of a Sprint iteration, at the beginning of which the set of user stories to work on are decided and which is shielded from interference for the duration of the sprint, is not ideal. By employing Kanban, students are allowed to finish up their project using an agile methodology that can adapt to their requirements.

4 EVALUATION

To understand students' perceptions of Kanban practices and to analyse whether the introduction of a new methodology successfully changed the workflow, we performed two types of evaluations: A survey at the very end of the course (before grades were announced) and an comparative analysis of the development artifacts produced during Kanban and Scrum project work.

4.1 Survey

We conducted a voluntary, anonymous online survey among all participants of the 2016/17 installment of the course. The survey of 11 questions focused on students' perceptions of Kanban practices, the biggest advantages and drawbacks and adaptations to the workflow that were performed. All questions of the survey are listed in Table 3.

Of the 22 attendees of the course, 18 (17 men, one woman) participated in the survey. In addition to questions that could be answered on a 5-point Likert scale (1: strong no, 2: no, 3: neutral, 4: yes, 5: strong yes), the survey included free text questions as well as a question featuring multiple choice options. They survey could be submitted without all questions being answered. However, all Likert-scale questions were answered by all participants. An overview of the answers to these questions is presented in Table 4.

Overall, students showed positive attitudes towards the inclusion of Kanban in the course. Both questions regarding the use of Kanban (questions 1a, 1b) were answered, on average, positively, with mean values of 4.08 and 3.75 respectively. Students stated that they had adapted their workflows for Kanban (question 2). It is encouraging that students were enthusiastic about including Kanban in the next iteration of the course as well (question 6). The only question, where answer mean as well as 10% trimmed mean were close not positive, see Figure 2, was on the topic of additional lectures (question 7). This is somewhat expected, due the the different learning styles of students, with some enjoying self-study and others more guidance through lectures.

#	Type	Question Text
1a	5-point scale	Was the Kanban week at project end more useful and productive then a last week of Scrum?
1b	5-point scale	Did the Kanban week at project end offer a good insight into other agile development methods?
2	5-point scale	Did you have to adapt your workflow for the Kanban week?
3a	free text	What were the biggest advantages of using Kanban in your team?
3a	free text	What were the biggest disadvantages of using Kanban in your team?
4	choice	How did user stories change from using Scrum to Kanban?
5a	free text	In the next iteration of the course, how long should Kanban be used?
5b	free text	In the next iteration of the course, when should Kanban be employed (e.g. in the beginning)?
6	5-point scale	Would you recommend using Kanban to the participants of next year's course?
7	5-point scale	Should there have been an additional lecture concerning Kanban?
8	free text	Own feedback, e.g. on best practices to be followed for the next course?

Table 3: Questions of the student survey performed at the end of the software engineering course.

#	Question Topic	Mean	Std. Dev.	10% Trim. Mean	Median	Range
1a	Kanban week preferred over another Scrum week?	4.08	1.38	4.30	5.00	4.00
1b	Kanban offered insight into agile methods?	3.75	1.29	3.90	4.00	4.00
2	Was the workflow adapted?	3.83	1.11	4.00	4.00	4.00
6	Recommended for next year?	4.33	0.98	4.50	5.00	3.00
7	Additional lecture on Kanban?	3.42	1.08	3.40	4.00	3.00

Table 4: Summarized answers of participants to the 5-point Likert scale questions of the survey. Answer possibilities: 1 (strong no), 2 (no), 3 (neutral), 4 (yes) 5 (strong yes).

The free text answers to questions 3a and 3b were manually tagged with the topics they included, the set of topics was iteratively refined after every question was processed. The results of these two questions are summarized in Figure 3.
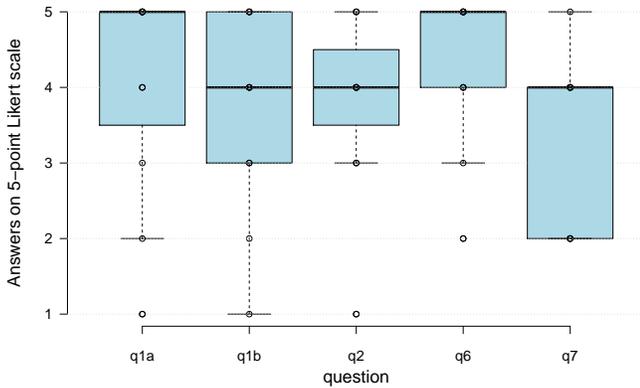
Figure 2: Boxplot of answers to questions 1a, 1b, 2, 6 and 7. Center lines show the medians, box limits indicate the 25th and 75th percentiles, whiskers extend 1.5 times the interquartile range from the 25th and 75th percentiles, outliers are represented by dots.
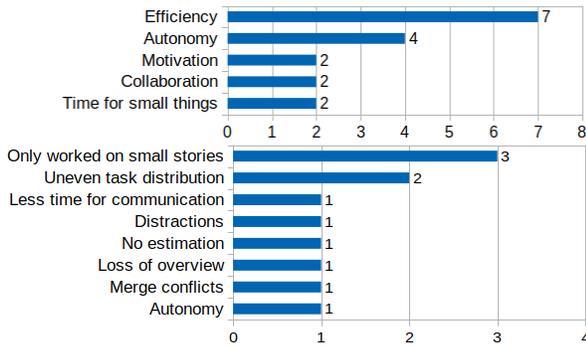

Figure 3: Topics of survey participants' answers to free text questions #3a and #3b, concerning the perceived advantages (top, N=11) and drawbacks (bottom, N=9) of using Kanban. An answer could contain multiple topics.

Concerning the perceived advantages of applying the Kanban method (question 3a, 3b), students most often mentioned the topics of efficiency and autonomy, concepts close to Lean Software's principles of *eliminating waste* and *empowering the team* [23]. The main drawbacks of Kanban as described by students in question 3b, were only working on small user stories (in favor of larger ones which were neglected) and uneven task distribution. Working mostly on small user stories can be seen as a consequence of autonomy, with developers choosing to work on small issues that can be finished quickly, instead of tackling large problems. Uneven task distribution is an ongoing challenge in self-organizing teams, which might be addressed in future by more thorough coaching of Scrum Masters and tutors to pay attention to this topic.

The main artifacts that developers and Product Owners use and communicate with, both in Kanban and Scrum, are user stories. Question 4 of the survey elicited how user stories changed from Scrum to Kanban. The multiple choice answer possibilities as well as a summary of answers are presented in Figure 4.
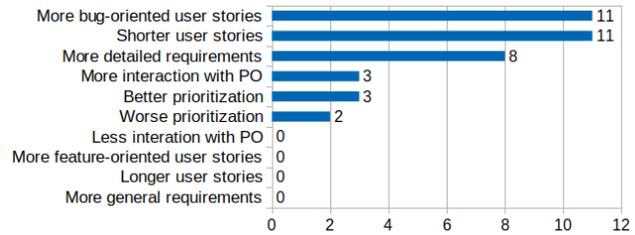
Figure 4: Answers of participants to survey question #4, concerning attributes of user stories when using Kanban compared to Scrum. N=12.

Preferred length of Kanban practice		
1 week	**1-2 weeks**	**> 2 weeks**
6	3	2

Preferred time for Kanban practice	
End of the project	**In the middle of the project**
9	1

Table 5: Summarized answers of students to the free text questions #5a and #5b concerning when and how long Kanban should be employed during the course. N=10.

As expected for Kanban usage at the end of a project, students classified the user stories that they wrote and worked on as shorter and more bug-oriented than they had previously been using Scrum. However, students also noted that user stories during Kanban contained more detailed requirements, in terms of acceptance criteria, a good attribute when trying to move tickets through the Kanban board as fast as possible. Concerning the best time for introducing Kanban in the course, students overwhelmingly agreed with our decision to employ Kanban towards the end of the project for a week, see Table 5.

The last question on the survey, question 8, asked for feedback on best practices to be followed for the next course iteration in the form of free text. This more reflective answer was only answered by 5 participants. However, the answers revealed areas that should be improved in future. Students pointed out that merges were painful and that they lacked a thorough understanding of merge strategies (3 mentions) as well as that instructors should focus on strategies of dealing with errors in others work and should highlight the importance of self-reliance (both 2 mentions). These are areas that will be addressed in future lectures and guest lectures by industry practitioners.

In summary, the survey revealed positive student attitudes towards the teaching approach of a shift from Scrum to Kanban at the end of the project work (*RQ2*), recommending the method to participants of the next year's course. Students stated that they had adapted their workflow (*RQ4*), which was intended as part of the learning experience and had gained insights into another agile method. They agreed that Kanban was well suited for the end of the project (rather than the beginning or the middle) and independently identified the advantages of Kanban over Scrum.

4.2 Development Artifact Analysis

While surveys allow insights into the perceptions of students, they do not tell the whole story. It is unclear, solely from survey answers, whether e.g. the perceived change in user story attributes or change in workflow actually happened and how severe the change was. In order to answer questions such as these, we performed an analysis of the development artifacts produced by course participants during Scrum and Kanban usage (*RQ3*). All main development activity around producing code and user stories, as tickets in an issue tracker, happened on GitHub, where the project was hosted. The service features comprehensive APIs[4], that allow querying both the issue tracker as well as the git version control system. For every development iteration, i.e. Sprints 1 to 4 as well as the Kanban week, all created user stories as well as commits were extracted and analyzed. Using this data, both the assumptions of learning success as well as the perceptions of students could be tested. Students' perception of shorter user stories while practicing Kanban is reflected in the data. While mean title length did not change significantly, the mean text body length of user stories was lower in the Kanban week, than in previous sprints, see top of Table 6. The dynamic interactivity with user stories on a Kanban board is reflected in the fact that in the Kanban week only 68.8% of user stories were created by POs, compared to much higher numbers in the Scrum Sprints, see bottom of Table 6.

However, the analysis also revealed that the uneven task distribution, mentioned by students in the survey, was also not fixed by employing Kanban: The number of unique assignees that worked on user stories did not significantly change and not all developers (19) were assigned a user story. The more dynamic nature and focus on throughput of Kanban was also apparent in analyses concerning commits. While diff sizes of commits in the Kanban week did not differ from those of previous Scrum Sprints (see top of Table 7), many more commits overall were made (see bottom of Table 7). Students' call for a more thorough introductions to helpful merge strategies is strengthened by the fact that the mean amount of merge commits almost tripled (from 51.5 to 142) from the last Scrum Sprint to the Kanban week.

The collection and evaluation of students' development data allowed another dimension of analysis, in addition to proven surveys. Development artifacts such as tickets and commits are necessarily produced during regular development activity and represent information on the executed process and possible adaptations to changes in development processes. This is especially relevant when comparing this tool to surveys, where not every participant of the course fills out the survey. However, almost every participant produces artifacts (and students that do not produce any are also of interest to educators). By comparing perceptions of students to what actually happened during the course, as read from development data, insights into those areas of the course that need improvement the most can be gained, e.g. the focus on teaching effective merge strategies. Using development data, supporting surveys, we were able to improve our assumptions on how students adapt their workflow when employing Kanban.

---

5 RELATED WORK

Related work for this paper, in addition to research into agile methods and their application (see Section 2) is found in the areas of research into how these methods can be taught in education as well as the domain of development process assessment.

5.1 Teaching Agile Capstone Courses

Many of the challenges in software development that agile approaches address are also found in the domain of student learning: complexity (introduction of entirely new concepts), under-defined problems (students are not familiar with problem space), time-boxed development with frequent team meetings (semesters and regular lectures), and inevitable change (applying new knowledge). Scrum can be viewed not only a software development method, but also as a general learning strategy [32]. Therefore, educating university students on agile processes using capstone courses, where these strategies can be applied first-hand, seems like a natural fit. In a literature review in 2015, Mahnič identified 23 primary studies on the topic of teaching Scrum in software engineering courses [14]. The author points out that all studies emphasized the need for Scrum to be taught taught using practical projects. Therefore, the usage of Scrum in capstone projects, requiring students to work in teams, was the most widely adopted strategy, described in seven studies [14]. More recent research in the context of agile capstone courses has focused on differences in Scrum usage between high and low performing teams, concluding that while the frequency of Scrum usage did not differ significantly, high performing teams applied Scrum practices more thoroughly [21].

Most capstone courses previously described in literature focus on the Scrum methodology. A systematic literature review in 2013 by Ahmad et al. on the application of Kanban in software development identified 19 relevant primary studies. Of these, however, none dealt with educational issues [2]. The same authors later conducted a study on student perceptions towards the software factory as a learning environment, where they reported the use of Kanban boards [1]. Mahnič et al., having identified the gap in the existing literature on using Kanban in software engineering education, describe a course design similar to the one proposed here. The authors divided the student teams of their Bologna master's program course into two groups: one using Scrumban and the other Kanban. Students were encouraged to experiment with Kanban and Scrum practices and to search for improvements in their software development process by focusing on the average lead time [13].

5.2 Evaluating Development Processes

There has been an ongoing debate on the compared effectiveness of Scrum or Kanban usage. Lei et al., in a study from 2017, note that their literature review found a lack of statistical evidence on the topic [12]. Based on the six project management factors of the Project Management Body of Knowledge (PMBOK 4.0), the authors performed a survey with 35 respondents to evaluate Scrum and Kanban in industry. While Lei et al. point out that Kanban may be better suited for managing project schedules, they conclude that based on their results, both Scrum and Kanban lead to the development of successful projects [12].

Iteration	# Issues	Title length			Text body length			Comment count			Events count		
		Mean	Stdev	Median	Mean	Stdev	Median	Mean	Stdev	Median	Mean	Stdev	Median
Sprint 1	10	45.7	12.2	46	630.4	487.3	526	1.5	2.59	0	21.2	3.5	21
Sprint 2	17	45.9	14.7	45	889.1	501.8	713	2.2	2.96	1	23.5	10.5	21
Sprint 3	31	38.2	13.6	35	804.7	405.9	768	1.5	1.86	1	18.2	9.5	17
Sprint 4	31	48.1	17.5	44	617.5	513.1	504	0.4	0.92	0	13.6	5.4	12
Kanban	32	34.8	17.2	30	315.7	367.4	221.5	1.2	1.77	1	7.7	3.6	7

Iteration	Opened by PO	Closed by PO	Same person open/close	Unique assignees count
Sprint 1	100%	60.0%	50.0%	9
Sprint 2	100%	29.4%	5.9%	12
Sprint 3	87.1%	12.3%	12.9%	18
Sprint 4	90.3%	41.9%	29.0%	15
Kanban	68.8%	6.3%	9.4%	16

Table 6: Summary of issues, i.e. user stories, by iteration. Events are all interactions with an issue, e.g. *labeled*, or *assigned*, except commenting on an issue.

Iteration	Files changed per commit			Insertions per commit			Deletions per commit		
	Mean	Median	Stdev	Mean	Median	Stdev	Mean	Median	Stdev
Sprint 1	8.79	4	15.5	109	17	282	58	6	200
Sprint 2	4.33	3	4.3	33	14	65	29	8	70
Sprint 3	4.33	2	7.5	139	9	2004	323	6	5952
Sprint 4	4.43	3	6.23	32	10	72	26	7	73
Kanban	3.67	2	4.47	37	10	69	25	6	50

Iteration	Mean per week				
	Non-merge commits	Merge commits	Files changed	Insertions	Deletions
Sprint 1	46	4.5	400	4948	2625
Sprint 2	74	23.5	321	2437	2153
Sprint 3	124	49.3	549	17580	122891
Sprint 4	138	51.5	619	4435	3664
Kanban	289	142.0	1069	10642	7320

Table 7: Commit statistics by iteration. Normalized by commit (top) and per week (bottom). Some iterations had different lengths.

In the domain of education, students' development processes are often evaluated in respect to the learning goals of the course. Mahnič notes capstone courses provide an opportunity to study student' perceptions of agile practices. He states that students behave similarly to young professionals in capstone courses, as these take place at the very end of their studies [14].

Development processes in capstone courses are generally evaluated in only a few different ways:

- Using surveys on students' perceptions of their executed process and the value of different dimensions involved in the project [3]
- Employing tutors that guide student teams and collect insights into teams [26]
- Taking end of term exams [8]
- Collecting and analysing the development artifacts produced during the course [15]

6 CONCLUSION

In this paper we introduce a university capstone software engineering course design, teaching agile methodologies and focusing on the hands-on practice of Scrum and Kanban. By first introducing the more prescriptive Scrum, with clearly defined meetings, roles and artifacts, students are free to experiment and get comfortable in their self-organized teams, while being able to rely on a given structure. Kanban can then be introduced to ease the inevitable phase of the project where the software is prepared for release while giving students a more thorough understanding of the agile landscape, beyond Scrum. Surveys at the end of the course revealed positive attitudes towards the course and the introduction of Kanban. Students stated that they endorsed the idea of using Kanban at the end of the course for a short period of time. An analysis of development data produced during the course, both during Scrum sprints as well as during Kanban application, corroborated the learning target of students having adapted their process to a new methodology.

REFERENCES

[1] Muhammad Ovais Ahmad, Kari Liukkunen, and Jouni Markkula. 2014. Student perceptions and attitudes towards the software factory as a learning environment. In *2014 IEEE Global Engineering Education Conference (EDUCON)*. IEEE, 422–428. https://doi.org/10.1109/EDUCON.2014.6826129

[2] Muhammad Ovais Ahmad, Jouni Markkula, and Markku Oivo. 2013. Kanban in software development: A systematic literature review. In *39th Euromicro Conference on Software Engineering and Advanced Applications*. IEEE, 9–16. https://doi.org/10.1109/SEAA.2013.28

[3] Maria Cecilia Bastarrica, Daniel Perovich, and Maira Marques Samary. 2017. What Can Students Get from a Software Engineering Capstone Course?. In *39th International Conference on Software Engineering: Software Engineering Education and Training Track (ICSE-SEET)*. IEEE, 137–145. https://doi.org/10.1109/ICSE-SEET.2017.15

[4] Denise A. Bonebright. 2010. 40 years of storming: a historical review of Tuckman's model of small group development. *Human Resource Development International* 13, 1 (feb 2010), 111–120. https://doi.org/10.1080/13678861003589099

[5] Georgia M. Kapitsaki and Marios Christou. 2014. Where is Scrum in the current Agile world?. In *ENASE 2014 - Proceedings of the 9th International Conference on Evaluation of Novel Approaches to Software Engineering*. 101–108. https://doi.org/10.5220/0004867701010108

[6] Henrik Kniberg. 2009. *Kanban and Scrum - Making the most of both.* 1–49 pages.

[7] Ayelt Komus and Moritz Kuberg. 2017. *Abschlussbericht : Status Quo Agile 2016/2017.* Technical Report. Hochschule Koblenz, University of Applied Sciences.

[8] Martin Kropp and Andreas Meier. 2013. Teaching agile software development at university level: Values, management, and craftsmanship. In *2013 26th International Conference on Software Engineering Education and Training (CSEE&T)*. IEEE, 179–188. https://doi.org/10.1109/CSEET.2013.6595249

[9] Martin Kropp and Andreas Meier. 2014. New Sustainable Teaching Approaches in Software Engineering Education How agile methods influence teaching software engineering. *2014 IEEE Global Engineering Education Conference (Educon)* April (2014), 1019–1022.

[10] Marco Kuhrmann, Philipp Diebold, Jurgen Munch, Paolo Tell, Kitija Trektere, Fergal Mc Caffery, Garousi Vahid, Michael Felderer, Oliver Linssen, Eckhart Hanser, and Christian Prause. 2018. Hybrid Software Development Approaches in Practice: A European Perspective. *IEEE Software* (2018), 1–1. https://doi.org/10.1109/MS.2018.110161245

[11] Corey Ladas. 2009. *Scrumban - Essays on Kanban Systems for Lean Software Development.* Modus Cooperandi Press, USA.

[12] Howard Lei, Farnaz Ganjeizadeh, Pradeep Kumar Jayachandran, and Pinar Ozcan. 2017. A statistical analysis of the effects of Scrum and Kanban on software development projects. *Robotics and Computer-Integrated Manufacturing* 43 (feb 2017), 59–67. https://doi.org/10.1016/j.rcim.2015.12.001

[13] Viljan Mahnic. 2015. From Scrum to Kanban: Introducing Lean Principles to a Software Engineering Capstone Course. *International Journal of Engineering Education* 31, 4 (2015), 1106–1116.

[14] V Mahnič. 2015. Scrum in software engineering courses: An outline of the literature. *Global Journal of Engineering Education* 17, 2 (2015), 77–83.

[15] Christoph Matthies, Thomas Kowark, Matthias Uflacker, and Hasso Plattner. 2016. Agile metrics for a university software engineering course. In *IEEE Frontiers in Education Conference (FIE)*. IEEE, Erie, PA, 1–5. https://doi.org/10.1109/FIE.2016.7757684

[16] Christoph Matthies, Arian Treffer, and Matthias Uflacker. 2017. Prof. CI: Employing continuous integration services and Github workflows to teach test-driven development. In *IEEE Frontiers in Education Conference (FIE)*. IEEE, 1–8. https://doi.org/10.1109/FIE.2017.8190589

[17] Natalja Nikitina, Mira Kajko-Mattsson, and Magnus Strale. 2012. From scrum to scrumban: A case study of a process transition. In *2012 International Conference on Software and System Process (ICSSP)*. IEEE, 140–149. https://doi.org/10.1109/ICSSP.2012.6225959

[18] Taiichi Ohno. 1978. Toyota Production System: Beyond Large-Scale Production. *Productivity Press* 1, 1 (1978), 152. https://doi.org/10.1108/eb054703 arXiv:arXiv:1011.1669v3

[19] Maria Paasivaara, Ville Heikkilä, Casper Lassenius, and Towo Toivola. 2014. Teaching Students Scrum using LEGO Blocks. *Companion Proceedings of the 36th International Conference on Software Engineering - ICSE Companion 2014* (2014), 382–391. https://doi.org/10.1145/2591062.2591169

[20] Maria Paasivaara, Casper Lassenius, Daniela Damian, Petteri Raty, and Adrian Schroter. 2013. Teaching students global software engineering skills using distributed Scrum. In *2013 35th International Conference on Software Engineering (ICSE)*. IEEE, 1128–1137. https://doi.org/10.1109/ICSE.2013.6606664

[21] Maria Paasivaara, Jari Vanhanen, Ville T Heikkilä, Casper Lassenius, Juha Itkonen, and Eero Laukkanen. 2017. Do High and Low Performing Student Teams Use Scrum Differently in Capstone Projects?. In *Proceedings of the 39th International Conference on Software Engineering: Software Engineering and Education Track (ICSE-SEET '17)*. IEEE Press, Piscataway, NJ, USA, 146–149. https://doi.org/10.1109/ICSE-SEET.2017.22

[22] Mia Persson, Ivan Kruzela, Kristina Allder, Olof Johansson, and Per Johansson. 2011. On the use of scrum in project driven higher education. *Proceedings of the World Congress in Computer Science, Computer Engineering and Applied Computing* (2011). http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.217.9885&rep=rep1&type=pdf

[23] Mary Poppendieck and Tom Poppendieck. 2003. *Lean software development: an agile toolkit.* Addison-Wesley.

[24] Ajay Reddy. 2015. *The Scrumban [R]evolution: Getting the Most Out of Agile, Scrum, and Lean Kanban.* Addison-Wesley Professional.

[25] David F. Rico and Hasan H. Sayani. 2009. Use of Agile Methods in Software Engineering Education. In *2009 Agile Conference*. IEEE, 174–179. https://doi.org/10.1109/AGILE.2009.13

[26] Andreas Scharf and Andreas Koch. 2013. Scrum in a software engineering course: An in-depth praxis report. In *26th International Conference on Software Engineering Education and Training (CSEE&T)*. IEEE, 159–168. https://doi.org/10.1109/CSEET.2013.6595247

[27] K Schwaber. 2004. *Agile Project Management with Scrum.* Vol. 7. Microsoft Press. 163 pages.

[28] Scrum Alliance. 2017. *The State of Scrum Report 2017 Edition.* Technical Report. Scrum Alliance. 31 pages. https://www.scrumalliance.org/scrum/media/ScrumAllianceMedia/FilesandPDFs/StateofScrum/State0fScrum_2016_FINAL.pdf

[29] The Joint Task Force on Computing Curricula. 2013. *Computer Science Curricula 2013: Curriculum Guidelines for Undergraduate Degree Programs in Computer Science.* ACM, Inc. 1–172 pages. https://doi.org/10.1145/2534860

[30] Georgios Theocharis, Marco Kuhrmann, Jürgen Münch, and Philipp Diebold. 2015. Is Water-Scrum-Fall Reality? On the Use of Agile and Traditional Development Practices. Lecture Notes in Computer Science, Vol. 9459. Springer International Publishing, Cham, 149–166. https://doi.org/10.1007/978-3-319-26844-611

[31] VersionOne Inc. 2017. *The 11th Annual State of Agile Report.* Technical Report. VersionOne Inc. https://explore.versionone.com/state-of-agile/versionone-11th-annual-state-of-agile-report-2

[32] Charles Wallace, Sriram Mohan, Douglas Troy, and Mark E. Hoffman. 2012. Scrum across the CS/SE curricula. In *Proceedings of the 43rd ACM technical symposium on Computer Science Education - SIGCSE '12*. ACM Press, New York, New York, USA, 5. https://doi.org/10.1145/2157136.2157142

[33] James P Womack, Daniel T Jones, and Daniel Roos. 1991. *The machine that changed the world: the story of lean production.* Harper Collins.