International Journal of Engineering Education Vol. 31, No. 4, pp. 1106–1116, 2015
Printed in Great Britain

0949-149X/91 $3.00+0.00

From Scrum to Kanban: Introducing Lean Principles to a Software Engineering Capstone Course*

VILJAN MAHNIČ
Faculty of Computer and Information Science, University of Ljubljana, Večna pot 113, 1000 Ljubljana, Slovenia.
E-mail: viljan.mahnic@fri.uni-lj.si

In this paper, a capstone course in software engineering is described that exposes students to lean principles advocated by Kanban. While retaining the main characteristics of its predecessor course, which concentrated on teaching agile software development using Scrum, the new course also introduces the most important Kanban concepts, i.e., visualization of the workflow and limitation of the work in progress. Kanban concepts are introduced in two ways: in combination with Scrum (as Scrumban) or as a "pure" Kanban (omitting some of the Scrum activities considered waste). Students are required to work in teams responsible for the implementation of a set of user stories defined by a project domain expert playing the role of the Product Owner. During the course, they must maintain a Kanban board and measure lead time. The paper discusses the use of different Kanban boards and work in progress limits, and analyzes the students' progress in reducing lead time. A summary of the lessons learned and recommendations is given reflecting the issues to be considered when teaching similar courses. A survey among students has shown that they liked both approaches and were overwhelmingly positive about the course.

Keywords: lean software development; Kanban; Scrumban; capstone course; software engineering education

1. Introduction

While agile methods [1] have gained wide acceptance among mainstream software developers, the use of lean principles [2] is a rather new but very promising approach paving its way into the software industry. In order to fulfill industry needs, teaching agile and lean software development has become an important issue when defining a software engineering curriculum. The Software Engineering 2004 Curriculum Model [3] did not pay enough attention to this issue, and until a few years ago courses on agile methods were rather rare [4]. The core software engineering courses mostly covered the traditional plan-driven approach (with rare exceptions; e.g., [5]); however, there is substantial evidence in the literature reporting the benefits of using the agile approach in capstone courses; for example [4, 6–8]. In the future, it is expected that the revision of the Software Engineering Curriculum Model will embrace the agile approach as a valid part of the curriculum from the start [9].

The most widespread agile method is Scrum [10]; according to the latest State of Agile Survey [11] Scrum and its variants are used by 73% of respondents. Therefore, substantial effort has been spent recently to find appropriate ways for incorporating Scrum within the scope of software engineering courses [12–14], study relationships between how students use Scrum and their learning style [15], and develop alternative approaches of teaching Scrum through educational games [16] and virtual reality support [17].

At the University of Ljubljana a Scrum-based software engineering capstone course was introduced in the academic year 2008/2009. After teaching the course for the first time, an empirical evaluation [18] revealed that students enjoyed learning Scrum and successfully grasped its main strengths, but lacked the abilities to estimate and plan, and did not fully understand the Scrum concept of a task or user story being "done". Therefore, the course was upgraded in the academic year 2009/2010 [8], with the project work being designed as an observational study providing data for empirical evaluation of some typical Scrum practices: user stories estimation [19], Sprint and release planning [20], and the usefulness of user stories for requirements specification [21]. Since that time, the aforementioned course design has been successfully used at the undergraduate level.

In the academic year 2013/2014, a decision was taken to upgrade the course with lean approaches to software development [2], in particular Kanban [22], and offer it as an elective course within the scope of the new Bologna master's program. The decision was motivated by some recent publications [23, 24] reporting significant improvements achieved by companies that adopted Kanban, as well as by the results of the last two State of Agile Development surveys [25, 11], indicating the rapid growth of Kanban and Scrumban users in industry. In [23], the case of a Scandinavian company is described, which after the introduction of Kanban almost halved the lead time, reduced the number of weighted bugs by 10 percent, and improved pro-

* Accepted 18 January 2015.

ductivity. Similarly, [24] reports experience from a Microsoft maintenance project indicating that the typical lead times, from the arrival of a request to its completion, were reduced from 125–155 days to only 14. According to [25], the number of Kanban and Scrumban users nearly doubled in 2012 and further increased in 2013 [11].

While Kanban has been used in manufacturing for many years [26], it is a relatively new concept in the area of software engineering. It can be used in combination with Scrum as the so called Scrumban [27], although the advocates of "pure" Kanban claim that some Scrum practices (e.g., fixed-length Sprints, user stories estimation, Sprint and release planning meetings, strict prioritization of user stories in the Product Backlog) should be abandoned since they represent waste not adding customer value directly. Kniberg and Skarin [28] stress that Kanban is less prescriptive than Scrum; therefore, Kanban users are expected to experiment with the process in order to customize it to their environment. A systematic literature review [29] identified only 19 studies on Kanban usage; however, none of these studies dealt directly with educational issues. Ikonen et al. [30] used students as subjects in order to explore the sources of waste in Kanban software development projects and only recently the use of Kanban board in project-based courses was reported in a study of student perceptions and attitudes towards the software factory as a learning environment [31].

The aim of this paper is to fill this gap by describing the content of the upgraded software engineering capstone course at the University of Ljubljana and the experience gained after teaching it for the first time. The remainder of the paper is organized as follows: Section 2 describes the overall course design and project setting in the academic year 2013/2014. In Section 3 the Kanban board structures and work in progress (WIP) limits used are explained in more detail. Section 4 describes lead time measurement results, while Section 5 presents the results of a post-course survey. Section 6 discusses the most important lessons learned and the recommendations that may help teachers who plan to teach a similar course. Section 7 provides a conclusion.

2. Course description

On the basis of the positive experience with the previous Scrum-based course [8], it was decided to retain its main characteristics that proved useful, i.e., realistic simulation of professional experience through team-work on a quasi-real project, the use of user stories for requirements specification, an explicit role of the Product Owner, strict enforcement of the notion of "done", and empirical evaluation of students' performance. On the other hand, it was decided to pay special attention to issues, which seem to be most important when introducing Kanban to software development, i.e., the structure of the Kanban board, assignment of WIP limits, and measuring lead time. Additionally, it was decided to experiment with different ways of introducing Kanban: a stepwise introduction through Scrumban or "pure" Kanban approach abandoning some of the Scrum practices.

2.1 Overall design

The aim of the course is to teach agile and lean software development through practical work by augmenting Scrum with the lean concepts of Kanban. It is assumed that the students have already mastered traditional methods of software development, fundamentals of data bases and information systems in previous courses, as well as that they have some basic knowledge of software project management and agile methods, in particular Scrum. The majority of students who attended the course in the academic year 2013/14 have also completed the aforemntioned Scrum-based course at the undurgraduate level.

The course lasts one semester (15 weeks) and is divided into two parts. The first three weeks consist of formal lectures on Scrum, Kanban and how to apply user stories for requirements specification. These three weeks are also used to prepare the development environment and acquaint students with the initial Product Backlog containing a set of user stories [32] they are going to develop. The Product Backlog is developed by a project domain expert playing the role of the Product Owner. If the project is carried out within the Department, the Product Owner is one of the instructors; if the project is developed for an industry partner, this role is played by a representative of a company. Each user story should consist of a short description, which is used for planning and as a reminder for conversations, and acceptance tests, which serve for determining when a story is "done".

The rest of the course consists of practical project work. Students are divided into teams of four responsible for the development of the required functionality. In order to further explore the differences between Scrumban and "pure" Kanban, student teams are divided into two groups: the Scrumban group and the Kanban group.

2.1.1 Scrumban group

Teams belonging to the Scrumban group (in the remainder Scrumban teams) retain the Scrum concept of fixed-length iterations; therefore, the rest of their course is divided into three Sprints, each lasting four weeks. Each Sprint starts with a

Sprint planning meeting at which the contents of the next iteration is agreed with the Product Owner and the initial version of the Sprint Backlog is defined. Considering the rules of agile planning [33], the stories in the Product Backlog are prioritized and each team is required to (i) define its expected velocity and (ii) estimate user stories in story points using predefined specific values of 0.5, 1, 2, 3, 5, 8, 13, and 20. The content of the Sprint Backlog must not exceed the estimated velocity. At the end of each Sprint, the Sprint review and Sprint retrospective meetings are organized. At the review, the Scrumban teams present their results to the Product Owner, while at the retrospective meeting, students and instructors meet to assess the development process in the previous Sprint, giving suggestions for improvements in the next.

The aforementioned Scrum framework is augmented by the use of the Kanban board and WIP limits. The board visualizes the workflow, while the WIP limits prevent the team members from working on several work items at the same time, thus minimizing lead time. The Kanban board includes the Sprint Backlog column, which is initiated at each Sprint planning meeting.

2.1.2 Kanban group

Teams belonging to the Kanban group (in the remainder Kanban teams) follow lean concepts more strictly by abandoning fixed-length iterations and Sprint planning. They are no longer required to maintain the Sprint Backlog and track their velocity. Instead, the Product Owner maintains a small number of high priority stories, which a team member can pull into development whenever he/she completes the user story he/she worked on before. In order to ensure continuous workflow, the Product Owner is also expected to evaluate user stories promptly, as soon as each user story is signaled as finished.

Consequently, the review meetings are not held at regular intervals, but are event driven. A review meeting is triggered whenever there is a set of minimum marketable features (MMF), defined by the Product Owner, ready for release.

There is no difference between both groups regarding Daily Scrum and Sprint retrospective meetings. Teams belonging to the Kanban group hold their retrospective meetings regularly at the same intervals as Scrumban teams (i.e., at the end of each Scrumban Sprint) and all teams are required to meet regularly at the Daily Scrum meetings. However, since students cannot be expected to work on the project every day, Daily Scrum meetings are mandatory only twice per week.

The differences in the operation of Scrumban and Kanban teams are summarized in Table 1.

2.2 Project setting in 2013/2014

In the academic year 2013/2014, the course was taken by 66 students who were divided into 16 teams. Two teams consisted of 5 students. During the course 4 students dropped out, so that the course was successfully completed by 62 students. The teams were randomly split into halves: eight teams were required to develop their projects using Scrumban, while the other eight had to use "pure" Kanban. In order to additionally strengthen the learning of Kanban concepts and their use in practice, the project consisted of developing a Web-based tool for managing Kanban projects. The teacher played the role of Product Owner who maintained the Product Backlog, while three teaching assistants helped him in answering questions on details of user stories, monitoring the development process, and evaluating work being done. A free version of Kanbanize project management tool (https://kanbanize.com/) was used to maintain the Kanban board.

The initial Product Backlog consisted of 18 user stories and was further augmented during the project execution in order to simulate changes in user requirements and priorities. Altogether, there were 24 user stories to be implemented (10 "must have", 6

Table 1. The differences between Scrumban and Kanban teams

Concepts used	Scrumban	Kanban
Iterations	Fixed-length Sprints.	Continuous workflow (no iterations).
Sprint planning meetings	Held regularly, at the beginning of each Sprint. Sprint Backlog must be defined and maintained.	No Sprint planning meetings. Product Owner maintains a small subset of high priority stories.
Sprint review meetings	Held regularly, at the end of each Sprint.	Event driven when there is a set of MMF ready for release.
Sprint retrospective meetings	Held regularly, at the end of each Sprint.	Held regularly, at the same time as Scrumban groups.
Daily Scrum meetings	Held regularly; however, due to the students' other obligations only twice per week. Kanban board is used to monitor workflow.	Held regularly; however, due to the students' other obligations only twice per week. Kanban board is used to monitor workflow.

"should have", and 8 "could have") describing the required functionality for four different user roles: the Kanban Master, the Product Owner, the Team Member, and the System Administrator. The role of Kanban Master was analogous to the role of Scrum-Master in Scrum. Assuming that the Kanban Master is responsible for methodology, the tool was envisioned to offer him/her the possibility to define and adapt the structure of the Kanban board, prescribe the WIP limits, and monitor progress through cumulative flow diagrams and lead time calculations. With regard to the Product Owner, the tool was required to enable him/her to define work items in the form of user stories and to decide when a user story is done. The Team Members were supposed to have the possibility to estimate work items and move them through development from one workflow state to another. The System Administrator was assumed to assign each user his/her role(s) and maintain the data required for the proper functioning of the system, i.e., data about developers, development teams, and projects they are working on.

The tool had to be as flexible as possible, giving the possibility to define a Kanban board with an arbitrary number of columns (representing different workflow states) and rows (representing different projects a development team can work on simultaneously). Each user had to be allowed to have several roles on the same project and play different roles in different projects. Special attention had to be devoted to the specification of rules for moving work items from one column to another. For each column, the tool had to permit the specification of who (i.e., which user role) can pull a work item from, as well as where the item can be moved. In order to compute the lead time, each move had to be assigned a time-stamp, thus making it possible to determine how long a work item remained in each workflow state.

Scrumban teams allocated the stories to Sprints strictly considering their priority and the estimated velocity of each team. For the purpose of event driven review meetings performed by Kanban teams, four sets of related stories were defined as MMF, each of them representing an increment of functionality that had to be reviewed and eventually deployed. The first set represented an internal release consisting of administrative operations required for proper functioning of the system (user administration, team definition, project attributes). The second set comprised core functionality required by Kanban (Kanban board maintenance, WIP limits, work items presentation, card moving from one workflow state to another). The third set concentrated on various analyses and reports required for development process monitoring (lead time calculation, cumulative flow diagram, WIP limit violations, deadline warnings). The fourth set included enhanced functionality, offering additional reports, customization of Kanban board and user stories display, as well as incorporation of ground rules of who (which role) can use the Kanban board and how.

Students' grades were determined on the basis of the amount of Product Backlog accomplished and the average lead time achieved, the quality of software and documentation developed, and the instructors' judgment on how well the team worked together, performed the prescribed meetings, and maintained the Kanban board.

3. Kanban board structure and WIP limits

During the project each team was required to maintain its Kanban board, which served as a visual control mechanism indicating how the work flowed through the various stages of the development process. The initial board structure was prescribed by instructors, but could later be adapted to better reflect the operation of each team.

3.1 Kanban board for Scrumban teams

It was suggested to the Scrumban teams that they use the board in Fig. 1. The "Product Backlog" column was intended to contain all user stories still waiting to be developed. Stories were prioritized by the Product Owner and estimated by each student team separately using planning poker or team estimation game. Whenever a new user story was created, a new card was added to this column.

The "Sprint Backlog" column contained user stories belonging to the current Sprint. The content of this column was initiated at the Sprint planning meeting when the Product Owner and the development team agreed which stories to develop in the next Sprint. During Sprint, the stories moved to subsequent columns in accordance with Kanban pull mechanism. If the Sprint was executed properly, this column became empty at the end of the Sprint and was filled again at the next Sprint planning meeting. The WIP limit of this column was expressed in terms of story points representing estimated velocity of each development team.

The rest of the board followed the idea from the literature [1, 34] that each iteration of an agile methodology is a self-contained, mini-project, with activities that span requirements analysis, design, implementation, test, and customer acceptance. Consequently, activities performed by student teams were united within the "Development" column consisting of the "Analysis & Design", "Coding", "Testing", "Integration", and "Documentation" subcolumns. The "Analysis & Design"

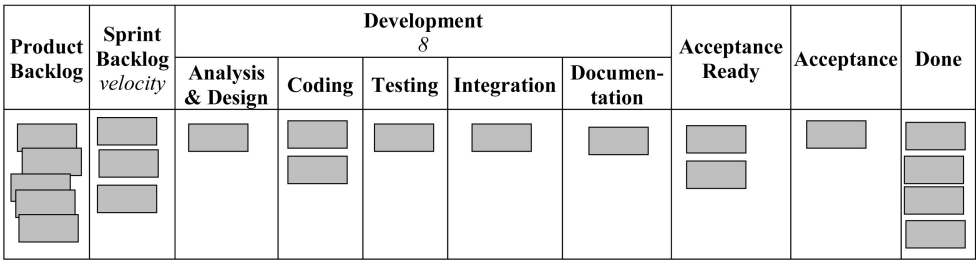

Fig. 1. Kanban board for Scrumban teams.

subcolumn was introduced to reflect the agile concept of just-in-time design and stress the need for clarifying each user story details through conversations with the Product Owner. The ''Coding'' subcolumn corresponded to coding and unit testing, while the ''Testing'' subcolumn represented functional testing. The ''Integration'' subcolumn was introduced to stress the requirement that all stories should be integrated into a working solution in order to be accepted by the Product Owner as ''done''. Experience from previous years had shown that, at the Sprint review meetings, far too frequently students wanted to present user stories in isolation not paying enough attention to consistent working of the application as a whole. The ''Documentation'' subcolumn reflected the Scrum requirement that the user operation of the new functionality developed in each Sprint must be documented, either in Help files or in user documentation [10].

The WIP limit of 8 was defined for the ''Development'' subcolumn as a whole, thus indicating that the total number of user stories in all its subcolumns should not have exceeded 8. A more detailed explanation of how WIP limits were defined follows in subsection 3.4.

The ''Acceptance Ready'', ''Acceptance'', and ''Done'' columns were introduced to stress the Scrum rule that it is the Product Owner who decides whether a story is ''done'' or not. The ''Acceptance Ready'' column served as a buffer between the development team and the Product Owner. Whenever a student team completed a user story, they moved the corresponding card to the ''Acceptance Ready'' column, thus giving the Product Owner a sign to start acceptance testing. Then the Product Owner pulled the card into the ''Acceptance'' column and, if the user story passed all acceptance tests, moved it to the final column ''Done''. Otherwise, the card was moved back to the ''Sprint Backlog'' and its color was changed to indicate that the story was rejected and needed some rework.

Considering the fact that (theoretically) a development team can submit all stories for evaluation at the end of the Sprint, the WIP limits for ''Acceptance Ready'' and ''Acceptance'' columns were not explicitly defined. Consequently, the amount of work in these columns was only limited implicitly by the size of the Sprint Backlog. Nevertheless, the Product Owner made every effort to evaluate each user story promptly, as soon as the corresponding card appeared in the ''Acceptance Ready'' column.

3.2 Kanban board for Kanban teams

The board structure of Kanban teams differed in the second column; instead of the ''Sprint Backlog'' the ''Next'' column was introduced as shown in Fig. 2. The ''Next'' column was intended to contain a limited number of high priority stories, which the Product Owner wanted to be implemented first. The WIP limit of 4 was chosen because there were 4 students in each development team. Whenever one of them was ready to start working on a new item, he/she could take a user story from the ''Next'' column and move it to ''Analysis & Design''. This was a sign to the Product Owner to choose the next story with the highest priority from the ''Product Backlog'' and fill up the free space in ''Next''. Within the WIP limit of the ''Next'' column, the Product Owner was allowed to change priorities by

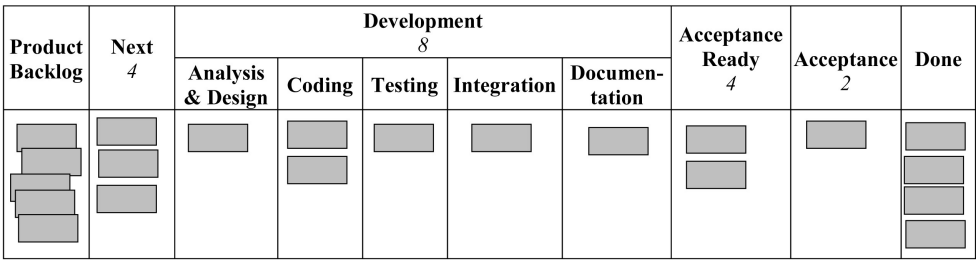

Fig. 2. Kanban board for Kanban teams.

moving high priority stories from "Product Backlog" to "Next" and vice versa. If the "Next" column contained 4 stories, one of them had to be moved back to the "Product Backlog" before being replaced with a more urgent user story.

WIP limits were also introduced in the "Acceptance Ready" and "Acceptance" columns in order to check for possible bottlenecks in acceptance testing and ensure that acceptance testing was performed as soon as possible after the completion of each story. The WIP limit of 4 in the "Acceptance Ready" subcolumn allowed, on average, one story per team member to wait for evaluation at any given moment. On the other hand, the WIP limit of 2 in the "Acceptance" column allowed the Product Owner to test at most 2 user stories simultaneously.

3.3 Adaptation of the board structure

During the project student teams were encouraged to adapt, if necessary, the board structure to their actual development process. Fig. 3 shows an example of how the "Development" column was adapted to better fit the needs of those teams that strictly separated responsibilities for developing the back end and front end of each user story. Apart from splitting the Coding subcolumn into "Back end" and "Front end", the "Ongoing" and "Completed" subcolumns were introduced to distinguish between user stories being still in a working state ("Ongoing") and user stories waiting to be pulled into the next step in the process ("Completed").

By moving a card to the "Back end/Completed" subcolumn, the back end developers indicated that they had finished their part of the job and the front end development could start. Similarly, a card in the "Front end/Completed" subcolumn indicated that the story was ready to proceed to functional testing. WIP limits of "Back end" and "Front end" subcolumns ensured that the workload was balanced between both parts of the team.

3.4 Adaptation of WIP limits

Defining appropriate WIP limits is another important issue when introducing Kanban. A too high WIP limit may cause idle tasks without being worked on and consequently bad lead time. On the other hand, a too low WIP limit may prevent developers from starting new work, thus having as a consequence idle people and bad productivity. Again, there are no explicit rules what the WIP limits should be.

In order to expose students to this issue, the main idea was to start with a rather loose WIP limit and gradually tighten it until a bottleneck occurred. Accordingly, the initial WIP limit of the "Development" column was set to 8 (as shown in Figs. 1 and 2), which (on average) allowed each developer working on two user stories at the same time. The intention of this limit was to allow each developer to work regularly on one user story and, if necessary, also pull into development a story that had been rejected by the Product Owner. It was expected that this way of working would accelerate correction of rejected stories and consequently reduce lead time.

The first retrospective meeting revealed that the WIP limit was so generous that it had never been reached. On the contrary, it was misused by some students who pulled into development more than one new story at the same time. Therefore, the WIP limit of the "Development" column was reduced to 7 (which also appeared not to be too restrictive) and then to 5. At the same time, the WIP limits of 3 were introduced in the "Back end" and "Front end" subcolumns for teams using the board in Fig. 3. It was agreed that the WIP limits could only be violated if a special reason existed, which had to be recorded in the electronic project management tool.

The more severe WIP limits were useful for provoking congestions indicating possible bottlenecks in the development process. For example, a team that used the Kanban board in Fig. 3 reached the WIP limit of the "Back end" subcolumn several times because of cards waiting in the "Back end/Completed" subcolumn to be pulled into development of front end. In their case the front end part of the team appeared to be a bottleneck that was not able to handle quickly enough the work completed by the back end part of the team.

With the Kanban boards in Figs. 2 and 3, the WIP limit 5 in the "Development" column appeared to be too low towards the end of the course when some

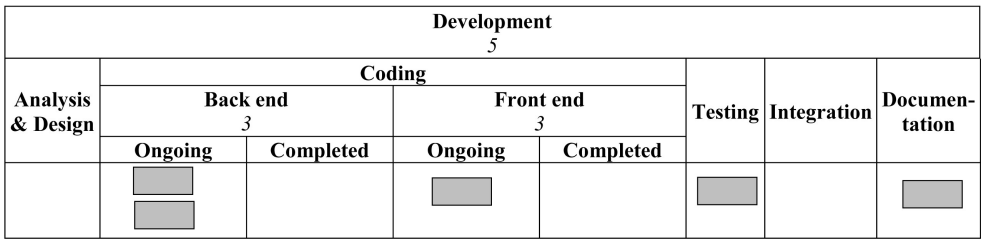

Fig. 3. Adaptation of the "Development" column to reflect work division between front end and back end.

late teams, apart from working on regular user stories, also had to resolve bugs in stories that had been rejected by the Product Owner. Towards the end of the course, when some teams intensified their work in order to complete as many as possible of the missing user stories, some violations of the WIP limit in the "Acceptance Ready" column were also noticed indicating that the Product Owner could not promptly check all user stories submitted for evaluation.

4. Measurement of lead time

At the second retrospective meeting, each team was required to report the average lead time for the stories they had already completed and provide proposals for its improvement. Lead time was expressed in terms of days that elapsed between the time a user story card was pulled into development (i.e., into the "Analysis & Design" subcolumn) and its arrival in the "Done" column.

Most proposals for improvement suggested stricter adherence to Kanban rules, in particular not pulling a user story into development until the work actually starts and not working on several items at the same time. Other proposals included increasing communication with the Product Owner, improving communication and work co-ordination among team members, and introducing pair programming and/or peer reviews of code. It was expected that better communication with the Product Owner would improve lead time by reducing the number of user stories that were rejected due to misunderstanding of user requirements. Improved communication and work co-ordination among team members was expected to reduce time needed for integration, while the introduction of pair programming and/or peer reviews of code was expected to increase the quality of developed programs. Additionally, the instructors encouraged the students to submit completed user stories for approval as soon as they were finished and not wait until the next lab practice hours, which were officially scheduled only once per week. For this purpose, additional contact hours were designated allowing students to meet the instructors daily, either for presenting results of their work or asking questions regarding user stories details.

At the third retrospective meeting, the average lead time was calculated again and compared to previous values. The great majority of teams reduced their average lead time, some of them for more than 30%. There were only two teams (T12 and T15, both from the Scrumban group) that did not succeed in reducing the average lead time, mostly due to some late user stories they started in Sprint 2 and completed in Sprint 3. Detailed results are shown in Table 2, representing (for each team separately) the number of completed user stories and the average lead time reported at the retrospective meetings 2 and 3, respectively.

Table 2. Number of completed user stories and the average lead time reported by each team at the retrospective meetings 2 and 3, respectively

	Retrospective meeting 2		Retrospective meeting 3	
Team	Stories completed	Average lead time (days)	Stories completed	Average lead time (days)
T01	24	8.85	24	8.85
T02	13	23.95	24	15.96
T03	17	12.73	24	10.60
T04	5	37.42	12	25.58
T05	8	24.71	18	23.76
T06	4	25.83	18	15.72
T07	15	12.13	24	9.17
T08	14	15.11	22	11.43
T09	4	31.28	15	21.81
T10	9	14.86	19	10.92
T11	24	7.21	24	7.21
T12	6	16.42	18	16.55
T13	16	12.06	24	10.31
T14	10	16.89	15	15.32
T15	6	16.50	20	18.05
T16	6	17.18	24	15.95

5. Students' evaluation of the course

In order to obtain students' feedback, an anonymous survey was conducted at the end of the course that consisted of two parts. The first part was devoted to a general evaluation of the course, while the second part dealt with students' satisfaction with the method they used. The survey was answered by 57 students out of 62 who completed the course.

5.1 General evaluation

The general evaluation of the course (see Table 3) clearly confirmed that students liked the course because of its orientation to learning through practical problem solving. The answers to question 1 show a unanimous support for the decision to teach the combination of agile and lean concepts through practical work on a quasi-real project. With regard to question 2, all students found the course either useful (40.4%) or useful and interesting (59.6%). The answers to question 3 also show that students were satisfied with the course. Almost half of them (42.1%) felt that the course was better than other courses and only two of them (3.5%) rated the course worse. Most of students (96.5%) also found the course beneficial to their employability and professional career (question 4).

The first part of the survey also contained two open-ended questions allowing students to specify what course aspects they liked most and least.

Table 3. Survey results — Part I: General evaluation of the course

1	Do you support the decision to teach agile and lean software development through practical work on a quasi-real project?		
	(a) yes	57	100.0%
	(b) no	0	0.0%
2	How useful is the course?		
	(a) the course is useful and interesting	34	59.6%
	(b) the course is useful	23	40.4%
	(c) the course is not useful	0	0.0%
	(d) the course is not useful and uninteresting	0	0.0%
3	How do you rate the course in comparison to other courses?		
	(a) better	24	42.1%
	(b) approximately the same	31	54.4%
	(c) worse	2	3.5%
4	Does the course contribute to your employability and professional career?		
	(a) yes	55	96.5%
	(b) no	2	3.5%

Almost all of them stressed practical work on a quasi-real project as the most positive experience. Among other things, they also mentioned teamworking, learning of state-of-the-art topics having industrial relevance, simulation of industrial environment and co-operation with a real customer. One of the students wrote: *"We actually saw how things are done in practice."*

On the other hand, the students found mandatory nature of the various kinds of meetings (mostly Daily Scrum, but also Sprint planning and retrospective) the most annoying part of the course. Some of them also complained about the amount of work required to complete all user stories in the project. However, it seems they were not right, since the best teams finished their projects before the end of the semester.

5.2 Satisfaction with the method used

The second part of the survey was intended to identify possible differences in students' opinions on Scrumban and Kanban. Students were asked first to rate the method they used on a scale from 1 to 5 (question 1), and then specify, which method they would have used for their project if they had been given the choice (question 2). Again, an open-ended question was added allowing them to additionally explain their answers to question 2. The results are presented in Table 4 for members of Scrumban and Kanban teams separately.

Both groups of students (Scrumban and Kanban) were satisfied with their method and, interestingly, the majority of them would choose the same method again. The satisfaction was greater among students who used Kanban. Their rate of Kanban was 4.12, and 84% of them would choose Kanban again. Students who used Scrumban rated their method slightly lower (3.84); 75% of them would use Scrumban again, while 25% would prefer to switch to Kanban.

Table 4. Survey results—Part II: Satisfaction with the method used

	Question	Scrumban (N = 32)	Kanban (N = 25)
1	Rate the method you used on your project	3.84	4.12
2	Which method would you use if you had a possibility to choose?		
	(a) Scrumban	24 (75%)	4 (16%)
	(b) Kanban	8 (25%)	21 (84%)

Among reasons for choosing Scrumban again, members of Scrumban teams put in the first place time-boxed iterations, because they clearly define deadlines for completion of work committed and provide more control over project execution. Many of them felt that time-boxed iterations encouraged them to work consistently rather than procrastinate. Another reason for preferring Scrumban over Kanban was their opinion that Scrumban is better because it combines the best of Scrum and Kanban. Several free-response comments illustrate the above findings:

- *"Scrumban prescribes more constraints, which enable better control over the project. Regular execution of prescribed meetings improves communication among team members, which I found beneficial for successful completion of our project. Due to fixed-length Sprints I always knew when my work should be finished."*
- *"Scrumban maintains regular cadence of Sprints with clearly defined deadlines, which encouraged me to work regularly without procrastination."*
- *"Regular Sprints help keeping sustainable pace of work."*
- *"Scrumban combines best practices of both methods."*

On the other hand, the members of Scrumban teams who expressed their preference for Kanban found Kanban better because of being less prescriptive, thus giving more freedom for adaptation and organizing work according to their needs. These students felt time-boxed iterations rigid and preferred event driven reviews of work completed. Their opinions were supported by comments like these:

- *"I prefer Kanban because it pays less attention to estimating and planning. No Sprint planning meetings are needed and the developers can concentrate on work that adds value to the customer."*
- *"I find Kanban more flexible and less bureaucratic."*
- *"By using Kanban it is easier to coordinate project work with commitments of other courses."*

Similar reasons were cited by those members of Kanban teams who would choose Kanban again. They also preferred Kanban because it is more flexible, less bureaucratic and requires fewer formalities (no Sprint planning meetings). Some of them mentioned event driven review meetings as a means for improving quality since the work completed is presented when it is really done and not because the end of a time-boxed iteration requires a presentation. Additionally, they found Kanban more appropriate because it allowed them to better combine their work on the project with other courses' commitments.

The members of Kanban teams who preferred Scrumban felt Scrumban better because of time-boxed iterations with fixed deadlines, which force developers to work consistently without procrastination.

6. Discussion

Introducing Kanban through team-project work has been a positive experience for both the teacher and the students. This section discusses the most important lessons learned that can contribute to improved course design and smoother running of the course in the future, as well as provide useful guidelines for teachers who are interested in teaching similar courses.

Teaching Kanban and lean concepts is an emergent issue in the area of software engineering education: Kanban is a relatively new concept in software engineering; however, positive experience of early adopters and the rapid growth of the number of its users indicate that introducing lean concepts can significantly improve software development. Therefore, teaching Kanban is becoming an important issue if we want to provide students with knowledge and skills needed in real life industrial projects. According to the author's experience, teaching Kanban is best done through a project-based course requiring students to apply lean concepts in practice, as well as to acquire a number of skills that cannot be gained solely through lectures and books.

Kanban can be successfully combined with agile methods like Scrum: In Kanban, the only constraints are Visualize Your Workflow and Limit Your WIP [28]. Therefore, the question arises how many additional rules should be imported from Scrum for successful Kanban implementation. In order to explore this issue in more detail, two approaches (i.e., "pure" Kanban and Scrumban) were used in the course. Scrumban required time-boxed iterations and regular execution of all Scrum meetings, while "pure" Kanban used no iterations and the review meetings were event driven. Both approaches proved to be successful and viable. Most students were satisfied with the approach they were required to use and the majority of them would use the same approach again if they were given the possibility to choose. Nevertheless, satisfaction with Kanban was slightly greater than satisfaction with Scrumban.

Defining appropriate structure of the Kanban board and WIP limits are principal issues when introducing Kanban: The structure of the Kanban board should closely reflect the actual development process used. By defining the initial board structure, instructors can impose on student teams the desired stages of the development process, especially those steps which students often do not pay enough attention to. On the other hand, students should be encouraged to search for the structure that best fits their development process. In the course described, a combination of both approaches was used. Some columns (e.g., "Testing", "Integration", and "Documentation") were prescribed because previous experience showed that students did not pay enough attention to these issues, while close monitoring of how particular teams work led to adaptations that improved visibility of the workflow and possible bottlenecks as in the example presented in Fig. 3. Adapting the board structure to their development process encourages students to think about the process and possible improvements. A possible solution, which should be explored in the future, is to ask each team to propose its own board structure, which should then be discussed and approved by instructors.

Experimenting with WIP limits gives students appropriate feeling of how WIP limits work and helps identifying possible bottlenecks: It is advisable to start with reasonably loose WIP limits, which should be gradually tightened to the point of causing congestions, thus helping to identify possible bottlenecks and discuss solutions for their removal. Ground rules should define what to do when a WIP limit is reached (e.g., allow violation of the limit if a special reason exists, or strictly prohibit it). Experience from the course has shown that the initial WIP limit of $2n$ (where n is the team size) for the "Development" column was too loose; therefore, it would be better to start with a more rigorous value (e.g., $3n/2$), since moving of rejected user stories to the "Next" and "Sprint Backlog" columns, respectively, did not affect WIP in the "Development" column. Additionally, the effects of limiting WIP in subcolumns of the "Development" column should be further explored in the future.

Measuring lead time strengthens the understanding of Kanban concepts and encourages improvements in the development process: Lead time is the main indicator of success in a Kanban implementation. It is recommended to require students to monitor

their average lead time and provide proposals how to improve it. Experience from the course has shown that obedience of Kanban rules (e.g., not to have more than one item per person in development at the same time) increased after the students had been required to measure lead time and provide recommendations for its improvement.

Prior knowledge of agile development methods, in particular Scrum, improves understanding of Kanban and Scrumban principles: In order to fully understand Kanban and Scrumbun principles, it is advisable to introduce Kanban after the students have already mastered agile software development methods, in particular Scrum. In our case the great majority of students got acquainted with agile methods and Scrum during their previous studies at the undergraduate level, which made it possible to start with ideas of Scrumban and Kanban from the very beginning of the course. This prerequisite should be considered as a limitation of applicability of this study and should be taken into account by teachers who plan to teach a similar course in the future.

7. Conclusion

The aim of this paper is to fill the gap in the existing literature on using Kanban in software engineering education. A detailed description of a software engineering capstone course is provided that exposes students to Kanban through practical team-project work. Apart from offering a new up-to-date content, the course gives students possibilities to explore different ways of combining Scrum and Kanban practices as well to experiment with different Kanban board structures and WIP limits. By monitoring the average lead time, they are encouraged to search for improvements in their software development process. A survey among students has shown that they were overwhelmingly satisfied with the course and found it beneficial to their employability and professional career.

In the future, it is planned to explore the sources of waste in students' projects and study the differences between Scrumban and Kanban in a more systematic manner. Regarding the first issue, it is envisioned that each student team will be required to maintain a value stream map of their development process clearly indicating how much time a work item spends in waiting states and non-value adding activities. The value stream map will have to be presented at each retrospective meeting and will serve as a basis for discussion on how to improve the development process and reduce lead time. Regarding the second issue, it is envisioned that students' projects will be carried out in a controlled environment that will allow formation of hypotheses about the strengths and weaknesses of both approaches. We expect that these results will also be useful for those companies that plan to introduce Kanban practices into their software development process.

Acknowledgments—The author thanks teaching assistants L. Fürst, M. Poženel, and A. Časar for their help in conducting the course, as well as all students who attended the course in the academic year 2013/2014 for their participation and useful comments, which made this work possible.

References

1. L. Williams, Agile software development methodologies and practices, *Advances in Computers*, **80**, 2010, pp. 1–44.
2. M. Poppendick and M. A. Cusumano, Lean software development: A Tutorial, *IEEE Software*, **29**(5), 2012, pp. 26–32.
3. Joint Task Force on Computing Curricula, Software engineering 2004: Curriculum guidelines for undergraduate degree programs in software engineering, *IEEE Computer Society and ACM CCSE, 2004*, http://sites.computer.org/ccse, accessed 6 September 2014.
4. D. F. Rico and H. H. Sayani, Use of agile methods in software engineering education, *Proceedings of the Agile 2009 Conference*, Chicago, USA, August 24–28, 2009, pp. 174–179.
5. L. Layman, L. Williams, K. Slaten, S. Berenson and M. Vouk, Addressing diverse needs through a balance of agile and plan-driven software development methodologies in the core software engineering course, *International Journal of Engineering Education*, **24**(4), 2008, pp. 659–670.
6. D. A. Umphress, T. D. Hendrix and J. H. Cross, Software process in the classroom: The capstone project experience, *IEEE Software*, **19**(5), 2002, pp. 78–85.
7. C. Coupal and K. Boechler, Introducing agile into a software development capstone project, *Proceedings of the Agile 2005 Conference*, Denver, CO, 2005, pp. 289–297.
8. V. Mahnič, A Capstone Course on Agile Software Development Using Scrum, *IEEE Transactions on Education*, **55**(1), 2012, pp. 99–106.
9. A. Fox and D. Patterson, Is the new software engineering curriculum agile?, *IEEE Softw*are, **30**(5), 2013, pp. 85–88.
10. K. Schwaber, *Agile Project Management with Scrum*, Microsoft Press, Redmond, WA, 2004.
11. VersionOne, 8th Annual State of Agile Survey, Atlanta, GA, 2014, http://www.versionone.com/pdf/2013-state-of-agile-survey.pdf, accessed 6 September 2014.
12. L. Werner, D. Arcamone and B. Ross, Using Scrum in a quarter-length undergraduate software engineering course, *Journal of Computing Sciences in Colleges*, **27**(4), 2012, pp. 140–150.
13. A. Scharf and A. Koch, Scrum in a software engineering course: an in-depth praxis report, *Proceedings of the 26th Conference on Software Engineering Education and Training (CSEE&T 2013)*, San Francisco, USA, 2013, pp. 159–168.
14. S. D. Zorzo, L. de Ponte and D. Lucredio, Using Scrum to teach software engineering: a case study, *Proceedings of the Frontiers in Education Conference*, Oklahoma City, USA, 2013, pp. 455–461.
15. E. Scott, G. Rodriguez, A. Soria and M. Campo, Are learning styles useful indicators to discover how students use Scrum for the first time?, *Computers in Human Behavior*, **36**, July 2014, pp. 56–64.
16. C. G. von Wangenheim, R. Savi and A. F. Borgatto, SCRUMIA–An educational game for teaching SCRUM in computing courses, *Journal of Systems and Software*, **86**(10), 2013, pp. 2675–2687.
17. G. Rodriguez, A. Soria and M. Campo, Virtual Scrum: a teaching aid to introduce undergraduate software engineering students to scrum, *Computer Applications in Engineering Education*, **23**(1), 2015, pp. 147–156.
18. V. Mahnič, Teaching Scrum through team-project work:

students' perceptions and teacher's observations, *International Journal of Engineering Education*, **26**(1), 2010, pp. 96–110.
19. V. Mahnič and T. Hovelja, On using planning poker for estimating user stories, *Journal of Systems and Software*, **85**(9), 2012, pp. 2086–2095.
20. V. Mahnič, A case study on agile estimating and planning using Scrum, *Elektronika ir Elektrotechnika* (*Electronics and Electrical Engineering*), **111**(5), 2011, pp. 123–128.
21. V. Mahnič and T. Hovelja, Teaching user stories within the scope of a software engineering capstone course: analysis of students' opinions, *International Journal of Engineering Education*, **30**(4), 2014, pp. 901–915.
22. D. Anderson, *Kanban—Successful Evolutionary Change for Your Technology Business*, Blue Hole Press, Sequim, WA, 2010.
23. D. I. K. Sjøberg, A. Johnsen and J. Solberg, Quantifying the effect of using Kanban versus Scrum: A case study, *IEEE Software*, **29**(5), 2012, pp. 47–53.
24. D. J. Anderson, J. Concas, M. I. Lunesu, M. Marchesi and H. Zhang, A comparative study of Scrum and Kanban Approaches on a real case study using simulation, in C. Wohlin (ed.): *XP 2012*, *Lecture Notes in Business Information Processing 111*, pp. 123–137.
25. VersionOne, 7th Annual State of Agile Development Survey, Atlanta, GA, 2013, http://www.versionone.com/pdf/7th-Annual-State-of-Agile-Development-Survey.pdf, accessed 6 September 2014.
26. T. Ohno, *Toyota Production System—Beyond Large-Scale Production*, Productivity Press, Portland, OR, 1988.
27. C. Ladas, *Scrumban—Essays on Kanban Systems for Lean Software Development*, Modus Cooperandi, Seattle, WA, 2008.
28. H. Kniberg and M. Skarin, *Kanban and Scrum—Making the Most of Both*, C4Media Inc., 2010.
29. M. O. Ahmad, J. Markkula and M. Ovio, Kanban in software development: a systematic literature review, *Proceedings of the 39th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA)*, Sep. 2013, pp. 9–16.
30. M. Ikonen, P. Kettunen, N. Oza and P. Abrahamsson, Exploring the sources of waste in Kanban software development projects, *Proceedings of the 36th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA)*, Sep. 2010, pp. 376–381.
31. M. O. Ahmad, K. Liukkunen and J. Markkula, Student perceptions and attitudes towards the software factory as a learning environment, *Proceedings of the Global Engineering and Education Conference (EDUCON)*, Istanbul, Turkey, Apr. 2014, pp. 422–428.
32. M. Cohn, *User Stories Applied for Agile Software Development*, Addison-Wesley, Boston, MA, 2004.
33. M. Cohn, *Agile Estimating and Planning*, Prentice Hall, Upper Saddle River, NJ, 2005.
34. C. Larman, *Agile and Iterative Development: A Manager's Guide*, Addison-Wesley, Boston, MA, 2004.

Viljan Mahnič is an Associate Professor and the Head of the Software Engineering Laboratory at the Faculty of Computer and Information Science of the University of Ljubljana, Slovenia. His teaching and research interests include agile software development methods, software process improvement, empirical software engineering, and software measurement. He received his Ph. D. in Computer Science from the University of Ljubljana in 1990.