World Transactions on Engineering and Technology Education *© 2015 WIETE*
Vol.13, No.3, 2015

The capstone course as a means for teaching agile software development through project-based learning

Viljan Mahnič

University of Ljubljana
Ljubljana, Slovenia

ABSTRACT: Since most of the core software engineering courses do not pay enough attention to agile software development, a software engineering capstone course represents an appropriate place for more in-depth treatment of this topic. In this article, the evolution of the software engineering capstone course at the University of Ljubljana is described since its conception in the 2008/09 academic year till now. The course requires students to develop a quasi-real project strictly following Scrum, which is the most widespread agile method. Additionally, the course design enables the conduct of studies that contribute to empirical evidence regarding agile processes. The article explains the reasons for introducing Scrum, presents the course design, describes some examples of empirical studies that were conducted within the course, and outlines the course upgrade with lean approaches to software development. Students' opinions about the course are overwhelmingly positive, indicating that the course is interesting and beneficial to their employability and professional career.

INTRODUCTION

In the last decade, agile development has moved from being a fringe approach to a widely employed software development methodology. In order to fulfil industry needs, teaching agile methods has become an important issue when defining a software engineering curriculum. The Software Engineering 2004 Curriculum Model did not pay enough attention to this issue and, until a few years ago, courses on agile methods were rather rare [1].

The core software engineering courses mostly covered the traditional plan-driven approach; therefore, a capstone course appeared to be an appropriate place for the introduction of agile software development. A capstone course not only offers the possibility to expose students to state-of-the-art topics having industrial relevance, but also stimulates the teacher to use modern ways of teaching agile processes through project-based learning. Additionally, conducting such a course as an observational study or incorporating controlled experiments can contribute to empirical evidence regarding new software processes that are constantly proposed for possible use in software engineering practice [2].

In light of the above, a software engineering capstone course was designed at the University of Ljubljana, Slovenia, that introduces Scrum as the most widespread agile method [3]. Students learn Scrum through team-project work requiring the development of a quasi-real project on the basis of user requirements provided by a domain expert playing the role of the Product Owner. Each student team plays the role of a Scrum team, which is collectively responsible for developing the required functionality.

The aim of this article is to present the evolvement of the course from its conception in the 2008/09 academic year till now. At the beginning, the main aim was to enrich the curriculum with industry-relevant topics, and special attention was devoted to students' perceptions about Scrum and its benefits [4]. Later on, the course design was upgraded with project work being designed as an observational study; thus, making it possible to combine teaching and research goals [5]. Data collected during the course served for an empirical evaluation of agile estimating and planning [6], an in-depth analysis of the planning poker estimation technique [7] and a complex analysis of students' opinions on user stories [8]. Finally, the course was upgraded with lean concepts of Kanban and Scrumban, and offered as an elective course within the scope of the new Bologna Master's programme [9]. Each of these steps will be illustrated in the subsequent sections.

INTRODUCING SCRUM AS A STATE-OF-THE-ART TOPIC

The University of Ljubljana was among the first universities that included agile methods into their curricula. In the 2008/09 academic year, a decision was taken to start teaching Scrum within the scope of a software engineering capstone course that the undergraduate computer science students take in the last semester of their studies. Scrum was introduced as a novel state-of-the-art topic that would attract students and increase their employability. It was expected that the students would prefer to learn Scrum through practical work than to follow classical lectures. Additionally, the project work should provide them with transferable skills that are necessary for a successful professional career.

For this purpose, two meaningful, practical projects were defined, one of them in co-operation with one of the largest software development companies in Slovenia. The teacher and a representative of the co-operating company played the roles of Product Owner of each project, while the students were divided into teams of four responsible for the implementation of functionality specified in the Product Backlog. All the work had to be done in two Sprints. For practical reasons, the teacher also played the role of ScrumMaster of all teams responsible for the Scrum process and for teaching Scrum to everyone involved in the project.

Since there was almost no experience with teaching Scrum reported in the literature, the course execution was closely monitored and special attention was devoted to data collection of students' perceptions in order to verify the course design and analyse student opinions on particular Scrum practices and artefacts [4].

Students' opinions were gathered through two questionnaires. The first questionnaire contained 10 questions on a 5-point Likert scale, with grade 1 representing the most negative, grade 3 representing neutral, and grade 5 representing the most positive opinion. The questions dealt with the use of Scrum and its practices: adequateness of the Product Backlog, accuracy of effort estimates, maintenance of the Sprint Backlog, administrative workload, co-operation with the ScrumMaster and Product Owner, quality of teamwork, appropriateness of workload, overall satisfaction with the work on the project, and overall satisfaction with the Scrum method. The questionnaire was answered at the end of each Sprint to evaluate students' experience for each iteration separately, as well as to analyse how students' opinions change when they get more experience.

Statistical analysis of students' responses revealed that 1) after the first Sprint, students' opinions were statistically significantly positive with regard to 7 questions out of 10; 2) after the second Sprint students' opinions were statistically significantly positive with regard to 9 questions out of 10; and 3) all grades except one increased after the second Sprint, four of them with a statistically significant difference. A significant positive correlation was found between students' satisfaction with the work on the project and the quality of the Product Backlog, maintenance of the Sprint Backlog, administrative workload, co-operation with the ScrumMaster, and co-operation with the Product Owner. These issues deserve most attention when introducing a similar course.

The second questionnaire served as a final survey at the end of the semester to obtain a general evaluation of the course as a whole, as well as for gathering students' opinions about the benefits of Scrum that are usually reported in the literature. The students unanimously supported the decision to run the course as a capstone project enabling them to learn Scrum by solving an almost-real problem.

The great majority of students (86.7%) found the course useful or useful and interesting. Almost half of them (46.7%) felt that the course was better than other courses in the final year of their study and only one rated the course worse. Most of the students (90.0%) also found the course beneficial to their employability and professional career.

In order to test the extent to which the students grasped the benefits of Scrum, their agreement was analysed according to nine assertions reported by Rising and Janoff [10]. For each assertion, the students were asked to specify how much they agree with using a 5-point Likert scale (1 - *strongly disagree* to 5 - *strongly agree*). Results reported in Table 1 revealed a statistically significant positive level of agreement with all assertions except assertion 2.

Table 1: Students' opinions regarding Scrum benefits.

		Mean	SD	One-sample *t*-test (*t*-value)
1	The product becomes a series of manageable chunks.	4.33	0.88	7.90**
2	Progress is made, even when requirements are not stable.	3.19	0.88	1.10
3	Everything is visible to everyone.	4.19	0.74	8.37**
4	Team communication improves.	4.04	0.90	6.00**
5	The team shares successes along the way and at the end.	4.59	0.57	14.46**
6	Customers see on-time delivery of increments.	4.07	0.78	7.15**
7	Customers obtain frequent feedback on how the product actually works.	3.93	1.00	4.83**
8	A relationship with the customer develops, trust builds and knowledge grows.	4.19	0.79	7.83**
9	A culture is created where everyone expects the project to succeed.	4.15	0.86	6.91**

$^{**}p < 0.01$

IMPROVED COURSE DESIGN

Feedback from the students and teacher's observations during teaching the course for the first time contributed to improved course design in the academic year 2009/2010 [5]. More attention was devoted to empirical evaluation of students' skills with special emphasis on agile estimating and planning [11]. A survey at the end of the course showed that students' opinions were even better than the year before. All students found the course useful or useful and interesting, 88% felt that the course was better than other courses in the final year of their studies, and 98% of them found the course beneficial to their employability and professional career.

As shown in Table 2, the course consists of Sprint 0 (a preparatory Sprint lasting 3 weeks) and three regular Scrum Sprints, each of them lasting 4 weeks. During Sprint 0 students are given formal lectures on Scrum and are presented with the initial Product Backlog consisting of user stories that must be implemented by the end of the course. The stories are written and prioritised by a domain expert (the teacher or a representative of a co-operating company) playing the role of the Product Owner. Students are grouped into teams of four responsible for the development of required functionality. Each team estimates the stories using planning poker and prepares the release plan.

Each regular Sprint starts with a Sprint planning meeting at which student teams define the contents of the next iteration and develop the initial version of the Sprint Backlog. During the Sprint, the teams have to meet regularly at Daily Scrum meetings and maintain their Sprint Backlogs, while at the end of each Sprint, the Sprint review and Sprint retrospective meetings take place. At the review, the students present their results to the instructors, while at the retrospective meeting, the students and instructors meet to review the work spent in the previous Sprint, giving suggestions for improvements in the next. After three Sprints, the first release should be completed and delivered to the customer.

Table 2: Course schedule.

Sprint 0	Weeks 1-3	Formal lectures on Scrum and user stories. Preparation of development environment. Initial Product Backlog presentation, user stories estimation, release plan development.
Sprint 1	Week 4	Sprint planning meeting, initial Sprint Backlog development.
	Weeks 4-7	Project work, Daily Scrum meetings, Sprint Backlog maintenance.
	Week 7	Sprint review meeting, Sprint retrospective meeting.
Sprint 2	Week 8	Sprint planning meeting, initial Sprint Backlog development.
	Weeks 8-11	Project work, Daily Scrum meetings, Sprint Backlog maintenance.
	Week 11	Sprint review meeting, Sprint retrospective meeting.
Sprint 3	Week 12	Sprint planning meeting, initial Sprint Backlog development.
	Weeks 12-15	Project work, Daily Scrum meetings, Sprint Backlog maintenance.
	Week 15	Sprint review meeting, Sprint retrospective meeting.
Release 1		Delivery of requested functionality, presentation of observational study results.

Students are required to provide data on their initial effort estimates, the amount of work spent and the amount of work remaining. At the beginning of each Sprint, they are encouraged to re-estimate their velocity and the remaining user stories in order to obtain a more realistic plan for future iterations. Instructors compare students' plans with actual achievement and analyse whether students' estimation and planning abilities improve as they gain more knowledge of Scrum and a better understanding of the user requirements. Special attention is devoted to the notion of *done*, requiring the students to bring the code up to the useful, real-world level, which can survive an encounter with end users.

COMBINING TEACHING AND RESEARCH GOALS

Besides providing students with a realistic software development experience, a capstone course in a software engineering programme offers considerable opportunities for more in-depth treatment of new topics from a research perspective. Conducting the course as an observational study or incorporating controlled experiments can contribute to empirical evidence regarding new software processes that are constantly proposed for possible use in software engineering practice [2][12]. Consequently, an appropriate combination of teaching and research goals can be beneficial for all parties involved: students, teachers, researchers and industry.

When teaching the course first started, the state of theory and research on Scrum was in the nascent phase [13]; therefore, one of the aims of the course was to help in filling this gap by conducting the course as an observational study. Moreover, the course design allowed the students to work on their own projects without loss of education time, and without being aware of research issues, and the actual purpose the collected data would be used for.

In the 2009/10 academic year, the course served as the basis for a case study on agile estimating and planning using Scrum [6]. The course was taken by 52 students who were divided into 13 teams that were required to develop a Web-based student records system covering enrolment, examination applications, examination records and some statistical surveys. Additionally, a special maintenance module had to be developed to enable the maintenance of all data required for the proper functioning of the system. The initial Product Backlog comprised 60 user stories and was the same for all teams. Fifty five stories described the required functionality for 4 different user roles (i.e. student records administrative staff, students, teachers and data administrator), whereas five stories described constraints that had to be obeyed. There were 24 *must have*, 5 *should have* and 4 *could have* stories required in the first release, which should have been completed in three Sprints by the end of the course. The remaining 27 *will not have this time* stories were specified merely to illustrate the desired functionality in the next release.

The aim of the study was to analyse students' abilities in estimating and planning; therefore, data on project management activities were collected in order to compare the results of different teams in terms of velocity (i.e. the amount of work completed in each Sprint), compliance with iteration plans, productivity and ability of effort estimation. A summary of results is presented in Table 3. Data clearly confirm the hypothesis that the plans are less accurate at the beginning, but improve from iteration to iteration.

Table 3: Comparison of planned and actual achievement across Sprints.

Sprint		Velocity [story points]		Plan fulfilment [%]	Work spent [hours]	Hours worked per story point
		Planned	Actual			
1	Mean	26.19	11.00	42.00	138.63	27.86
	Median	25.00	8.00	35.71	132.70	15.88
2	Mean	4.38	25.65	75.18	158.42	6.85
	Median	35.50	23.00	68.66	155.00	6.09
3	Mean	26.23	23.92	91.80	115.53	4.94
	Median	25.50	23.50	95.83	111.00	4.83

In the first Sprint, the actual velocity was far behind the planned (mean value 11.00, median 8.00). The teams completed on average only 42% (median 35.71%) of story points planned and spent on average much more than one ideal day of work per story point (mean value 27.86, median 15.88 hrs/story point). The most important reasons for such a great difference between plans and actual achievement were non-compliance with the concept of *done* and insufficient communication with the Product Owner on the part of students.

The difference between planned and actual achievement diminished significantly in the second Sprint. The actual velocity more than doubled and the teams completed on average 75.18% (median 68.66%) of story points planned. They spent on average 6.85 (median 6.09) hours per story point, which was almost in line with the concept of a story point being equal to 6 hours of work. Those teams that established good co-operation among team members, improved testing and integration, and delivered regularly user stories for evaluation, fulfilled their plans completely.

In the third Sprint, the teams estimated their velocity to be approximately the same as in the second Sprint, which proved to be the right decision (mean value 26.23, median 25.50). The actual achievement was very close to the plan (mean value 23.92, median 23.50). The teams completed on average 91.80% (median 95.83%) of story points planned and 5 teams achieved 100%. Two teams completed all the stories planned for the first release even before the end of the Sprint.

The results show that the ability to estimate and plan quickly improves. Most teams were able to define almost accurate Sprint plans after three Sprints. In the third Sprint, the velocity stabilised and the actual achievement almost completely matched the plan. Empirical data also showed a continued increase of productivity. These findings could be considered when introducing Scrum into industrial software development.

Another study concentrated on a comparison between effort estimates obtained through planning poker and effort estimates obtained as a simple average of individual estimates [7]. Scrum recommends the use of planning poker for estimating the size of user stories since it is assumed that the group discussion through planning poker helps in identifying activities that individual estimators could overlook and increases commitment of the team. Preliminary research has shown that planning poker reduces the over-optimism that is typical for expert judgment-based methods and provides more accurate estimates compared to statistical combination of individual estimates [14]. On the other hand, most studies in psychology and forecasting (e.g. see Buehler et al [15]) stress the possible hazards of group processes when predicting effort and schedule; thus, advising against group interaction.

In order to further evaluate the accuracy of planning poker estimates and compare students' estimates to estimates made by experienced professionals, data collected during the course execution were analysed to answer the following research questions:

- RQ1: How do planning poker estimates provided by student teams differ from the statistical combination of their individual estimates?
- RQ2: How do planning poker estimates provided by experienced professionals differ from the statistical combination of their individual estimates?
- RQ3: How do students' estimates differ from estimates made by experienced professionals?

At the end of Sprint 0, each team was required to estimate all user stories in story points using the planning poker technique. The estimates provided by each team member during the first round of planning poker were recorded and their statistical combination (the mean) was used for further comparison with the final planning poker estimate. The same stories were also estimated by a group of four experts in the problem domain. Again, the individual estimates of the first round and the final planning poker estimate of each story were recorded for the needs of the study. At the end of the course, the amount of work actually spent was compared to initial estimates.

With regard to research question RQ1, statistical analysis showed that both kinds of students' estimates tended to be over-optimistic. The use of planning poker did not reduce the over-optimism; on the contrary, the statistical combination of individual estimates provided by student teams proved to be less optimistic and more accurate than the planning poker estimates. With regard to experts (research question RQ2) statistical analysis revealed that their planning poker estimates tended to be more accurate than the statistical combination, but the difference was neither statistically significant nor had a substantial effect-size. With regard to research question RQ3, statistical tests revealed a significant difference in favour of experts for both kinds of estimates.

Comparison between students and experienced professionals helped to explain the mixed results of previous studies dealing with the effects of group discussion on estimation accuracy. The results of this study revealed that the optimism bias caused by group discussion diminishes or disappears as the expertise of people involved in the group estimation process increases. It seems that inexperienced software developers behave similarly to *ad hoc* groups performing estimates relating to tasks they are not familiar with (like in studies conducted by Buehler et al [15]), while highly motivated professionals are able to take advantage of group discussion to analyse the problem from different perspectives (like in the study conducted by Moløkken-Østvold et al [14]).

More recently, a complex analysis of students' opinions on user stories has been undertaken on the basis of several surveys that have been conducted among course participants since the 2009/10 academic year [8]. The analysis revealed that students' opinions significantly improve after they gain more experience; thus, confirming the hypothesis that project-based learning enables them to grasp the main concepts and understand the advantages and limitations of user stories.

PURSUING MODERN TRENDS

While Scrum and its variants are still the most widespread agile methods, lean principles advocated by Kanban are becoming increasingly popular. Early Kanban adopters report significant improvements in their development process (e.g. see Sjøberg et al [16]), and some experience with using Kanban in the academic environment has already been reported [17]. In order to pursue this trend, the course was upgraded with lean approaches and offered as an elective course within the scope of the new Bologna Master's programme in the 2013/14 academic year [9].

While retaining the main characteristics of its Scrum-based predecessor, the new course pays attention to issues that seem to be most important when introducing Kanban to software development, i.e., the structure of the Kanban board, assignment of work in progress (WIP) limits, and measuring lead time. 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 to be a waste). Consequently, student teams are divided into two groups: the Scrumban group and the Kanban group.

Teams belonging to the Scrumban group retain the Scrum concepts of Sprint planning and fixed-length iterations, while teams belonging to the Kanban group are no longer required to perform these activities. 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 signalled as finished. Consequently, the review meetings for Kanban group teams are not held at regular intervals, but are event driven. A review meeting is triggered whenever a set of minimum marketable features (MMF) is 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. Main characteristics of operation of Scrumban and Kanban teams are summarised in Table 4.

Table 4: Main characteristics of operation of 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. The 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 twice per week. Kanban board is used to monitor the workflow.	Held regularly twice per week. Kanban board is used to monitor the workflow.

During the project, each team is required to maintain its Kanban board, which serves as a visual control mechanism indicating how the work flows through the various stages of the development process. The initial board structure is prescribed by instructors, but can later be adapted to better reflect the operation of each team. An example of the Kanban board for Kanban teams is shown in Figure 1. Numbers in the Next, Development, Acceptance ready, and Acceptance columns represent the WIP limits that Kanban prescribes in order to maximise the workflow. During the course, students are encouraged to reduce the WIP limit in the development column in order to provoke congestions indicating possible bottlenecks in their development process. They are also required to report the average lead time and provide proposals for its improvement.

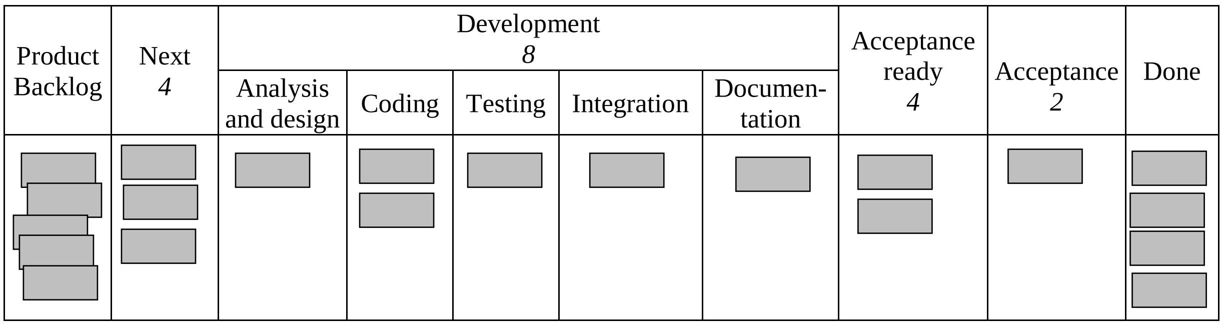

Figure 1: Kanban board for Kanban teams.

CONCLUSIONS

The experience of the University of Ljubljana was presented to illustrate different aspects that a software engineering capstone course offers for teaching agile methods. The course design not only allows the introduction of industry-relevant topics through project-based learning, but also makes it possible to conduct different studies that contribute to empirical evidence regarding new software processes without hindering the achievement of teaching goals. Students' opinions about the course are overwhelmingly positive and the course content is constantly being updated in order to follow modern trends in industry. The latest upgrade combines Scrum with lean concepts of Scrumban and Kanban. More details about the course can be found elsewhere [4-9].

REFERENCES

1. Rico, D.F. and Sayani, H., Use of agile methods in software engineering education. *Proc. Agile 2009 Conf.*, Chicago, USA, 174-179 (2009).
2. Carver, J., Jaccheri, L., Morasca, S. and Shull, F., Issues in using students in empirical studies in software engineering education. *Proc. 9th Inter. Software Metrics Symp.*, Sydney, Australia, 239-249 (2005).
3. Schwaber, K., *Agile Project Management with Scrum*. Redmond, WA: Microsoft Press (2004).
4. Mahnic, V., Teaching Scrum through team-project work: students' perceptions and teacher's observations. *Inter. J. of Engng. Educ.*, 26, **1**, 96-110 (2010).
5. Mahnic, V., A capstone course on agile software development using Scrum. *IEEE Trans. on Educ.*, 55, **1**, 99-106 (2012).
6. Mahnic, V., A case study on agile estimating and planning using Scrum. *Elektronika ir Elektrotechnika (Electronics and Electrical Engng.)*, 111, **5**, 123-128 (2011).
7. Mahnic, V. and Hovelja, T., On using planning poker for estimating user stories. *J. of Systems and Software*, 85, **9**, 2086-2095 (2012).
8. Mahnic, V. and Hovelja, T., Teaching user stories within the scope of a software engineering capstone course: analysis of students' opinions. *Inter. J. of Engng. Educ.*, 30, **4**, 901-915 (2014).
9. Mahnic, V., From Scrum to Kanban: introducing lean principles to a software engineering capstone course. *Inter. J. of Engng. Educ.*, 31, **4**, 1106-1116 (2015).
10. Rising, L. and Janoff, N.S., The Scrum software development process for small teams. *IEEE Software*, 17, **4**, 26-32 (2000).
11. Cohn, M., *Agile Estimating and Planning*. Upper Saddle River, NJ: Prentice Hall (2005).
12. Carver, J.C., Jaccheri, L., Morasca, S. and Shull, F., A checklist for integrating student empirical studies with research and teaching goals. *Empirical Software Engng.*, 15, 35-59 (2010).
13. Dingsøyr, T., Dybå, T. and Abrahamsson, P., A Preliminary roadmap for empirical research on agile software development. *Proc. Agile 2008 Conf.*, 83-94 (2008).
14. Moløkken-Østvold, K., Haugen, N.C. and Benestad, H.C., Using planning poker for combining expert estimates in software projects. *J. of Systems and Software*, 81, **12**, 2106-2117 (2008).
15. Buehler, R., Messervey, D. and Griffin, D., Collaborative planning and prediction: does group discussion affect optimistic biases in time estimation?. *Organizational Behavior and Human Decision Processes*, 97, 47-63 (2005).
16. Sjøberg, D.I.K., Johnsen, A. and Solberg, J., Quantifying the effect of using Kanban versus Scrum: a case study. *IEEE Software*, 29, **5**, 47-53 (2012).
17. Ahmad, M.O., Markkula, J. and Oivo, M., Kanban for software engineering teaching in a software factory learning environment. *World Trans. on Engng. and Technol. Educ.*, 12, **3**, 338-343 (2014).