A Capstone Course on Agile Software Development Using Scrum

Article *in* IEEE Transactions on Education · February 2012

DOI: 10.1109/TE.2011.2142311 · Source: IEEE Xplore

Viljan Mahnic
University of Ljubljana

A Capstone Course on Agile Software Development Using Scrum

Viljan Mahnic, *Member, IEEE*

Abstract—**In this paper, an undergraduate capstone course in software engineering is described that not only exposes students to agile software development, but also makes it possible to observe the behavior of developers using Scrum for the first time. The course requires students to work as Scrum 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, data on project management activities are collected in order to analyze the amount of work completed, compliance with the release and iteration plans, productivity, ability in effort estimation, and the like. The paper discusses the achievement of teaching goals and provides empirical evaluation of students' progress in estimation and planning skills. A summary of lessons learned and recommendations is given, reflecting the issues to be considered when teaching courses in agile software development. Surveys of students have shown that they were overwhelmingly positive about the course, indicating that the course fully met or even exceeded their expectations.**

Index Terms—**Agile software development, capstone course, effort estimation, Scrum, software engineering education.**

I. Introduction

RECOMMENDATIONS for undergraduate software engineering curricula developed jointly by the IEEE Computer Society and the Association for Computing Machinery (ACM) mandate that students undertake a capstone project, which covers one full year [1], [2]. The course should integrate previously learned material, deepen their understanding of that material, extend their area of knowledge, and apply their knowledge and skills in a realistic simulation of professional experience [3]. Additionally, the course can help students in developing transferable skills that are better acquired within a practical project framework than through formal lectures [4]–[6].

A. Agile Methods in Capstone Courses

Due to the increasing use of agile methods in industry in the last few years, the teaching of agile methods for software development [7], [8] has become an important issue [9]. Although some universities are starting to teach courses on agile methods, these are still rather new [10]. Since core software engineering courses mainly teach the traditional plan-driven approach, the capstone course seems to be an appropriate place for the introduction of agile software development.

Manuscript received October 05, 2010; revised December 02, 2010 and January 31, 2011; accepted March 17, 2011. Date of publication May 12, 2011; date of current version February 03, 2012.

The author is with the Faculty of Computer and Information Science, University of Ljubljana, Ljubljana 1000, Slovenia (e-mail: viljan.mahnic@fri.uni-lj.si).

Digital Object Identifier 10.1109/TE.2011.2142311

An extensive study of 49 capstone projects [11] using different software processes [ad hoc, MIL-STD-498, IEEE 1074, Team Software Process, and Extreme Programming (XP)] reports several benefits of introducing the agile approach in place of the traditional plan-driven processes. The agile approach is more appealing to student teams, and project success, measured by how well the finished software met customers' expectations, surpassed that of previous projects using nonagile software processes. Coupal and Boechler [12] report a similar experience comparing student projects developed following an agile approach to their previous projects developed in a traditional way. Successful introduction of agile methods in capstone courses is also reported in several other studies, for example [10] and [13]–[15].

While the results reported in the literature are mostly positive, some studies describe findings that are more in favor of the traditional plan-driven approach. For example, an observational study conducted by Robillard and Dulipovici [16] indicated that a disciplined process paradigm would be the more efficient approach for capstone projects for inexperienced students in software engineering.

This sometimes contradictory experience reported in the literature can be explained by the fact that the current state of theory and research on particular agile development methods is still in the nascent phase [17]. Dybå and Dingsøyr's systematic review of 33 empirical studies on agile software development [18] revealed that most of them investigated almost exclusively XP. Similarly, most experience in teaching the agile approach is based on the use of XP and its practices. On the other hand, a recent survey [19] on the state of agile development has shown that Scrum [20], [21] is the most widespread agile method in industry. Consequently, there is an increasing need to place more emphasis on teaching Scrum in order to prepare students for their professional careers.

B. Software Engineering Capstone Course at the University of Ljubljana

In light of the above, a new software engineering capstone course was designed at the University of Ljubljana, Ljubljana, Slovenia, that introduces Scrum as a framework for the planning and management of students' projects. The course ran for the first time in the academic year 2008–2009 and, as well as teaching students agile software development through teamwork on two industry-relevant projects, also paid special attention to students' perceptions about the learnability and usability of Scrum. An empirical evaluation [22] has shown that students enjoyed learning Scrum and successfully grasped its main strengths. Their perceptions of Scrum improved from Sprint to Sprint and were in statistically significant agreement with anecdotal evidence about Scrum's benefits reported in the literature. On the other hand, close monitoring of the student teams' performance revealed that students lacked the abilities to estimate and plan and did not fully understand the Scrum concept of a task or user story being "done."

In order to help students to better understand the problem of agile estimation and planning and to improve their skills, the course was upgraded in the academic year 2009–2010, with the project work being designed as an observational study providing data for empirical evaluation of students' skills with special emphasis on user stories estimation, release planning, fulfillment of scope, and velocity tracking. The upgraded course was designed to maximize the benefits for both students and the teacher as proposed by Carver *et al.* [23], [24]. Besides exposing students to state-of-the-art topics having industrial relevance, the upgraded course design stimulates the teacher to use modern ways of teaching agile processes through practical problem solving and provides empirical evidence of what students have learned, which can serve as a basis for short- and long-term course improvements.

The remainder of this paper is organized as follows. Section II contains an overview of the new course design. Section III describes in more detail the capstone project setting in the academic year 2009–2010. Section IV presents the results of the pre- and post-course surveys and discusses the achievement of teaching goals. Section V analyzes how students' estimation and planning abilities improved through iterations. Section VI outlines the most important lessons learned and recommendations that may help teachers who plan to teach a similar course.

II. Course Design

The course is designed to teach Scrum in a near real-world environment, augmenting the Scrum method with user stories, release planning, and velocity tracking [25]. Students are required to work in groups in order to develop a quasi-real project on the basis of user requirements provided by a domain expert playing the role of the Product Owner. The course lasts 15 weeks and is taken by computer science students in their last (eighth) semester. It is allocated 7 European Credit Transfer and Accumulation System (ECTS) points. Therefore, the expected workload of each student (including contact hours) is between 175 and 210 hours.

Since the students have already mastered the traditional methods of software development, the only formal lectures are in the first three weeks of the course to teach them Scrum and how to apply user stories for requirements specification and project planning. These three weeks (called Sprint 0 since they serve as a preparatory Sprint before the start of the project) are also used to prepare the development environment and acquaint students with the initial Product Backlog containing a set of prioritized user stories for the project they are going to develop. Students are grouped into teams of four responsible for the development of the required functionality. Each team estimates the stories using planning poker [26] and prepares the release plan.

The rest of the course is divided into three Sprints, each lasting four weeks as shown in Table I. Strictly following the Scrum method, each Sprint starts with a Sprint planning meeting at which student teams negotiate the contents of the next iteration with the Product Owner and develop the initial version of the Sprint Backlog. During the Sprint, the teams have to meet regularly at the Daily Scrum meetings and maintain their Sprint Backlogs, adding new tasks if required and updating data on work spent and work remaining. 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, students and instructors meet to review the work done in the previous Sprint, giving suggestions for improvements in the next. After three Sprints the first release should be complete and delivered to the customer.

TABLE I
Course Schedule

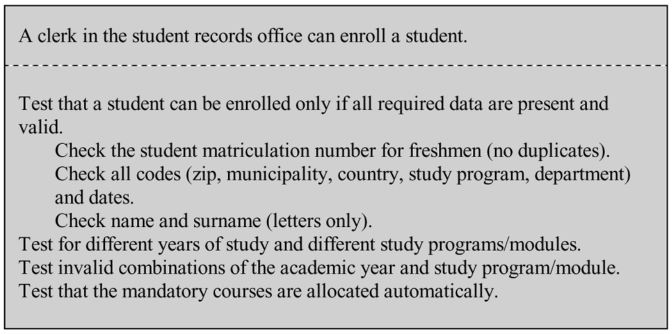
Sprint 0	Weeks 1-3	Formal lectures on Scrum and user stories. Preparation of development environment. Initial Product Backlog presentation, user stories estimation, release plan development.
Sprint 1	Week 4	Sprint planning meeting, initial Sprint Backlog development.
	Weeks 4-7	Project work, Daily Scrum 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 reestimate their velocity and the remaining user stories in order to obtain a more realistic plan for future iterations. Instructors compare students' plans to actual achievement and analyze whether students' estimation and planning abilities improve as they gain more knowledge of Scrum and a better understanding of the user requirements. An upgraded version of the Agilo for Scrum project management tool [27] is used to support the collection of all required data.

There is no formal final exam, and the students' grades are determined on the basis of the amount of Product Backlog accomplished, the quality of software and documentation developed, the fulfillment of release and Sprint plans, and the instructor's judgment on how well the team worked together, maintained the Sprint Backlog, and kept to schedule.

III. Project Setting in 2009–2010

In the academic year 2009–2010, the course was taken by 52 students divided into 13 teams. The project consisted of


A clerk in the student records office can enroll a student.

Test that a student can be enrolled only if all required data are present and valid.
  Check the student matriculation number for freshmen (no duplicates).
  Check all codes (zip, municipality, country, study program, department) and dates.
  Check name and surname (letters only).
Test for different years of study and different study programs/modules.
Test invalid combinations of the academic year and study program/module.
Test that the mandatory courses are allocated automatically.

Fig. 1. Sample user story.

developing a Web-based student records system covering enrollment, 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 (i.e., the maintenance of various code tables, lists of required and optional courses, data on teachers of each course, and so on).

Having had extensive experience of leading the development of the University of Ljubljana student records information system [28], and having been the Assistant Dean for Educational Affairs for several years, the teacher was able to define realistic user requirements and play not only the role of ScrumMaster, but also the role of Product Owner. Although these two roles are separated in real Scrum projects, this solution proved to be beneficial for two reasons: 1) the teacher (in the role of Product Owner) could work together with student teams as a collocated user representative providing timely answers to questions on user stories and quick evaluations of their implementation; and 2) in the role of ScrumMaster, he could ensure that all teams followed Scrum principles and practices and provided all data required for empirical evaluation of the development process.

During Sprint 0, student teams were presented with the initial Product Backlog consisting of 60 user stories. Fifty-five stories described the required functionality for four different user roles (student records administrative staff, students, teachers, and data administrator), whereas five stories described constraints that had to be obeyed (e.g., the system had to enable remote access to data through the Internet, all outputs should also be printable, and so on). Each story contained a short description and a set of acceptance tests that had to be used to demonstrate that the story had been correctly and fully coded. A sample story is shown in Fig. 1.

The stories were divided into four groups on the basis of priority. There were 24 "must have," 5 "should have," 4 "could have," and 27 "won't have this time" stories. The students were asked to implement all "must have" and as many as possible "should have" stories for the first release that could be used in practice. If they planned to be more productive, they could also include the "could have" and even some of the "won't have this time" stories. However, the "won't have this time" stories were specified merely to illustrate the desired functionality for the next release.

TABLE II
General Evaluation of the Course in Academic Years 2008–2009 and 2009–2010

Question	2008–2009		2009–2010	
	N=30	%	N=51	%
How useful is the course?				
a) the course is useful and interesting	21	70	36	71
b) the course is useful	5	17	15	29
c) the course is not useful	3	10	0	0
d) the course is not useful and is uninteresting	1	3	0	0
How do you rate the course in comparison to other courses in the final year of your study?				
a) better	14	47	45	88
b) approximately the same	13	43	6	12
c) worse	1	3	0	0
d) no response	2	7	0	0
Does the course contribute to your employability and professional career?				
a) yes	27	90	50	98
b) no	3	10	1	2

In order to prepare the release plan, each team estimated the stories in story points, using the planning poker estimation technique. A story point was treated as an ideal day of work (i.e., a day with no interruptions whatsoever), and the estimates were constrained to specific predefined values of 0.5, 1, 2, 3, 5, 8, 13, and 20. Considering their expected velocity, the teams then allocated stories into each of the three Sprints.

At the beginning of each Sprint, the teams had to define the Sprint Backlog, breaking the stories down into constituent tasks and assigning responsibility for each task. Each student individually estimated how many hours it would take to accomplish each task he/she had accepted.

During the Sprint, Daily Scrum meetings were held twice per week since students could not be expected to work on the project every day. At the Daily Scrum meeting, each team member had to record the number of hours spent on, and the amount of work remaining for, each task for which he/she was responsible. When the team finished a story, the Product Owner was asked to evaluate its implementation. The Product Owner strictly rejected all stories that did not conform to user requirements. If the shortcomings were not removed by the end of the Sprint, a new story was defined in the Product Backlog requiring completion of the missing features in one of the remaining Sprints.

Stories that had not been begun, or that were either rejected or newly defined by the Product Owner, were reestimated at the beginning of each consecutive Sprint in order to create a more realistic plan for subsequent iterations. At the end of the first Sprint, a substantial difference was noticed between the planned and actual velocity. In subsequent Sprints, the deviations from the plan were much smaller, and at the end, the majority of teams implemented all "must have" and "should have" stories, whereas the best team also accomplished all "could have" and some "won't have this time" ones.

IV. Achievement of Teaching Goals

A survey at the end of the 2009–2010 course showed that students' opinions were overwhelmingly positive (see Table II)

TABLE III
RESULTS OF THE FIRST PART OF THE SURVEY ($N = 51$): QUESTIONS 1 AND 2

Question		Median		p-value (WSR test)
		At the beginning	At the end	
1	Experience in the area of industrial software development	3	4	< 0.001
2	Familiarity with software development tools	3	4	< 0.001

TABLE IV
RESULTS OF THE SECOND PART OF THE SURVEY ($N = 51$): QUESTIONS 3–10

Question		Median		p-value (WSR test)
		Expected improvement	Actual improvement	
3	Familiarity with agile approach to software development	4	5	0.023
4	Knowledge of computer programming	3	4	0.234
5	Software projects planning and management skills	4	4	0.977
6	Effort estimation skills	4	4	0.682
7	"Big picture" about the software development process	4	4	0.087
8	Team-work skills	4	4	0.475
9	Customer interaction skills	4	4	0.002
10	Communication skills	3	4	0.002

and even better than the very good opinions of the year before. All students found the course either useful (29%) or useful and interesting (71%); 88% felt that the course was better than other courses in the final year of their study, and nobody rated the course as worse. All students except one (98%) also found the course beneficial to their employability and professional career.

In order to pursue the achievement of teaching goals in more detail, an additional survey was developed that was conducted at the beginning and at the end of the course. The survey consisted of two parts and contained 10 questions. The first part (Questions 1 and 2) was intended to evaluate: 1) the students' experience in the area of industrial software development, and 2) their familiarity with software development tools before and after the course.

The purpose of the second part (Questions 3–10) was twofold: to identify which skills students expected to improve the most during the course, and to analyze to what extent the course actually fulfilled these expectations. Identifying this at the beginning of the course was helpful in fine-tuning the course content, while students' opinions of the fulfillment of their expectations served as a measure of their satisfaction with the course. Students were asked to rate their expectations for, and their perception of, their improvement in the following skills (number refers to question in the survey): 3) familiarity with the agile approach to software development; 4) knowledge of computer programming; 5) software project planning and management skills; 6) effort estimation skills; 7) gaining a "big picture" of the software development process; 8) teamwork skills; 9) customer interaction skills; and 10) communication and presentation skills.

For Questions 1 and 2, students rated their experience and knowledge at the beginning and at the end of the course using a 5-point Likert scale with grade 1 representing no experience/knowledge and grade 5 representing considerable experience/knowledge. The 5-point Likert scale was also used to respond to Questions 3–10. However, at the beginning of the course, the students were asked to rate their expectations for the improvement of a particular skill ("How much do you expect the course will improve each of the following skills?"), while at the end of the course, they rated the actual improvement ("How much did the course actually improve each of the following skills?"). Grade 1 denoted no improvement, and grade 5 denoted the maximal improvement. Results on the basis of 51 responses are presented in Tables III and IV. Wilcoxon's signed ranks test (WSR test) was used to analyze the significance of the before–after effect.

The results in Table III clearly show that the course completely fulfilled the general objectives of a capstone project, significantly increasing the students' experience in the area of industrial software development (Question 1) and their familiarity with software development tools (Question 2).

Table IV compares the expected and actual improvement of different skills. The median values of the actual improvement after the course either match or exceed the median values of the expected improvement at the beginning, thus indicating that the students' expectations regarding improvement were met or even surpassed. In the statistical tests, the difference between the actual and expected improvement was being compared, identifying three skills with a statistically significant improvement over the students' expectations at the beginning of the course (see Questions 3, 9, and 10). The other skills, except for effort estimation, also improved more than expected, but not to a statistically significantly extent.

V. ASSESSMENT OF STUDENTS' PROGRESS IN ESTIMATION AND PLANNING SKILLS

Conducting the practical part of the course as an observational study under tightly controlled conditions made it possible to study the behavior of student teams using Scrum for the first time. This section provides an overview of data collected and analyzes students' progress in estimation and planning skills with regard to the following questions: 1) How does the ability to plan improve through iterations? 2) How does the velocity change from iteration to iteration? 3) How accurate are initial estimates?

A. *Ability of Planning*

The mean values of planned, reported, and actual velocity of 13 student teams are presented in Table V. The "Reported" column gives all stories that the teams reported as having been done at the end of each Sprint, while the "Actual" column only gives the stories accepted by the Product Owner. The data confirm the hypothesis that the accuracy of plans improves from iteration to iteration.

In the first Sprint, all teams except one overestimated their velocity, completing on average only 42% of story points planned. They also spent much more time per story point than defined in

TABLE V
COMPARISON OF PLANNED, REPORTED, AND ACTUAL ACHIEVEMENT

Sprint	Velocity (Story points)			Plan fulfillment (%)		Work spent (hours)	Hours worked per story point
	Planned	Reported	Actual	Reported	Actual		
1	26.19	22.96	11.00	87.92	42.00	138.63	27.86
2	34.38	30.81	25.65	89.86	75.18	158.42	6.85
3	26.23	24.50	23.92	93.70	91.80	115.53	4.94

Agilo, which assumes that a story point is worth 6 ideal hours of work. This finding is in line with the results of other studies using students as subjects [29], as well as with the findings of many human judgment studies showing that people tend to be overoptimistic when predicting their own performance [30]. The inaccurate plans were not only a consequence of students' inexperience in estimating and planning; other factors also contributed to the low actual velocity and, consequently, the high number of hours worked per story point. Because the system was new, some teams spent a substantial amount of time developing the infrastructure and getting acquainted with development tools, although the required infrastructure was expected to be in place by the end of Sprint 0. Some teams also spent more time than necessary refactoring the design of the underlying database. Misinterpreting the agile principle of minimal up-front design, the data model was often designed on the spot for each user story separately instead of considering the requirements of all stories in the Sprint together. Uncoordinated design resulted in a substantial amount of rework during integration.

It is also worth noting the great difference between the velocity reported by students and the actual velocity computed by the Product Owner. The Sprint retrospective meeting revealed that the difference between the reported and actual achievement was a consequence of noncompliance with the concept of "done" and insufficient communication with the Product Owner on the part of students. Most stories that students considered completed were rejected, either for not being robust enough to survive the encounter with the end-user or for not fully matching the user requirements.

The Product Owner's strict insistence on providing fully tested, integrated, and usable code was quite a surprise for some teams, who underestimated the differences between the trivial programming projects of their previous academic experience and the actual level of effort required for production class software required in a real-world environment. Consequently, they were recommended to pay more attention to activities that contribute to increased robustness and stability, such as testing, continuous integration, and configuration management.

With regard to the stories that did not fully meet user requirements, some teams, instead of being aware that the requirements details should be worked out in conversations with the Product Owner, complained that the noncompliance was due to the requirements details not being described precisely enough in the user stories. Therefore, students were strongly encouraged to increase their communication with the Product Owner during the subsequent Sprints. It was agreed that the Product Owner would be available for questions not only during the hours officially scheduled for lessons and lab practice (as in the first Sprint), but daily during specially designated contact hours and by e-mail. To resolve possible disagreements over the implementation as soon as possible, it was also agreed that all teams should submit their user stories for review as soon as they were completed rather than waiting until the Sprint review meeting.

A considerable improvement was noticed in the second Sprint. Student teams completed on average 75.18% of the story points planned, and three teams out of 13 fulfilled their plans completely. The difference between reported and actual achievement diminished due to more intensive communication between the teams and the Product Owner and prompt evaluation of completed user stories. More intensive communication increased mutual understanding of user requirements, while prompt evaluation of completed user stories enabled most of the disagreements to be discovered and resolved before the end of the Sprint. The completion of a story point required on average 6.85 hours of work, which was much less than in the first Sprint and almost in line with the concept of a story point being equal to 6 hours of work.

By strictly following the improvements agreed upon after the first Sprint, the initial problems and learning curves were, to a great extent, overcome, and most teams started to gather and sustain momentum that resulted in more user stories being implemented and accepted by the Product Owner. It became evident that those teams that established good internal cooperation, improved testing and integration, and regularly delivered user stories for evaluation achieved better results. It was therefore agreed at the Sprint retrospective meeting to further improve teamwork, acceptance testing, and continuous integration.

These improvements contributed to still better results in the third Sprint. Student teams completed on average 91.80% of the story points planned, and five teams achieved 100%. Two teams completed all the stories planned for the first release even before the end of the Sprint. The difference between the reported and actual velocity almost vanished. The average amount of work required for implementation of a story point was reduced to 4.94 hours, thus indicating a continued improvement of productivity. Inadequate internal communication and lack of teamwork remained the main reasons for the weaker performance of those teams that did not fulfill their plan.

B. *Velocity Tracking*

In the first Sprint, the planned velocity estimates were overoptimistic (mean value 26.19), partly because of the students' inexperience in estimating and planning and partly because they were unaware of the requirements imposed by the concept of "done" as discussed in the Section V-A. Only one team out of 13 actually reached the planned velocity. The actual velocity of all other teams was far behind that planned (mean value 11.00).

In spite of the experience from the first Sprint and the teacher's warnings not to be overoptimistic, surprisingly most teams increased their planned velocity for the second Sprint (mean value 34.38). It seemed that the teams wanted to make up for the deficit in the first Sprint and were blind to other opinions. Of course, their approach did not work. The planned velocity was again too high, in spite of the fact that the actual velocity more than doubled (mean value 25.65) compared to the first Sprint.

TABLE VI
COMPARISON OF ESTIMATED AND ACTUAL EFFORT

Team	Effort (Story points)		Absolute error	Relative error (%)
	Estimated	Actual		
T01	57.50	90.50	33.00	36.46
T02	64.50	81.17	16.67	20.53
T03	42.50	39.25	-3.25	-8.28
T04	30.50	48.67	18.17	37.33
T05	55.00	54.95	-0.05	-0.09
T06	37.00	55.02	18.02	32.75
T07	38.50	67.50	29.00	42.96
T08	38.50	47.83	9.33	19.51
T09	78.00	48.17	-29.83	-61.94
T10	51.00	55.50	4.50	8.11
T11	56.50	66.50	10.00	15.04
T12	62.50	55.92	-6.58	-11.77
T13	41.00	47.96	6.96	14.51

In the third Sprint, the teams finally recognized that the best way to plan their velocity was to draw on their experience from previous Sprints. They estimated their velocity to be approximately the same as in the second Sprint (mean value 26.23), and the actual achievement (mean value 23.92) was much closer to the plan.

The observation of the actual velocity of student teams confirmed the anecdotal evidence reported in the literature [25] that: 1) the velocity increases in the beginning and stabilizes after a couple of Sprints, and 2) the velocity is best estimated on the basis of historical values.

C. *Comparison of Estimated and Actual Effort*

Table VI shows the difference between estimated and actual effort considering only the "must have" and "should have" user stories that described functionality (not constraints) and were fully implemented by all teams. The "Estimated" column represents the sum of estimates defined by each team at the end of Sprint 0, while the "Actual" column represents the actual effort in story points. Negative values in the "Absolute error" and "Relative error" columns indicate that the actual effort was less than estimated.

Most teams (nine out of 13) underestimated the effort. Three teams' estimates were very close to actual effort (relative error less than 10%), while one team overestimated the effort significantly and needed less time than planned. The mean relative error for all 13 teams was 11.16%, and the mean magnitude of the relative error was 23.79%, which was quite good considering that an acceptable level for the mean magnitude of relative error is less than or equal to 25% [31].

Again, some students claimed that the estimates would have been more accurate if the user stories had been more detailed. However, specifying more details would have been a violation of the basic user story concepts. Therefore, the initial estimates depended merely on the Product Owner's ability to present stories and the students' abilities to understand requirements. They were refined during subsequent Sprints as students gained a deeper understanding of the requirements.

VI. Lessons Learned and Recommendations

1) Teaching Agile Methods Is Best Done Through Projects and Practical Work: Scrum itself is simple in concept, yet difficult to implement. Therefore, there is no need for extensive formal lectures to introduce the concepts. Instead, projects are the vehicle that assemble all the disparate issues affecting the implementation and allow these to be traded off against one another. However, students require adequate prior knowledge of the philosophy and techniques of software engineering and information systems development in order to properly evaluate the benefits and shortcomings of the agile approach compared to traditional plan-driven software development. In pursuing the challenge of combining formal with practical learning, the design of the capstone course can rely on previous courses having provided formal lessons on the aforementioned topics, so just a short theoretical introduction to agile methods (including the reason for their importance) is enough to encourage students' buy-in before starting the practical work.

2) Role of the Product Owner Is Crucial for the Success of a Scrum Project: He/she communicates the vision of what is to be developed and defines the criteria by which it will be judged. In order to assure smooth running of the course, it is important that the Product Owner provides timely answers to questions on details of user stories and makes quick evaluations of work being done. A nonresponsive Product Owner can cause unproductive work periods, which make iteration planning more difficult or even impossible. Additionally, the Product Owner must be knowledgeable enough about Scrum to be able to write, maintain, and prioritize the user stories. Otherwise, the ScrumMaster must help him/her prepare and maintain the Product Backlog. If the role of the Product Owner is played by the teacher, he/she must behave like a user representative, thus keeping the situation in the classroom as realistic as possible.

3) ScrumMaster Largely Performs Guardian Tasks: Besides teaching Scrum to everyone involved in the project and removing impediments, he/she must ensure that everyone obeys Scrum rules and practices. It is particularly important that he/she prevents the Product Owner from interfering in the management of teams, redefining the scope or goals of a Sprint, or trying to add new requirements once a Sprint has started. This makes it advisable to separate the roles of Product Owner and ScrumMaster. If, however, the teacher is an expert in the project domain and strictly respects the difference between the two roles, as in the experience described here, combining both roles is not problematic.

4) Notion of "Done" Should Be Imparted to Students as Soon as Possible: Strict enforcement of the concept of "done" contributes to the awareness that the code must be fully tested and resistant to user errors before being used in practice. Students all too often produce solutions that notionally offer the required functionality, but that lack robustness, reliability, and professional polish. The overoptimistic estimates and the large gap between the planned and the actual velocity at the beginning of the project were to a great extent the consequence of students underestimating the effort needed to bring the code up to the useful, real-world level that can survive an encounter with users.

5) Concepts of User Stories Are Best Imparted to Students Through Practical Work, but Only if Good Communication Is

Established Between the Product Owner and the Team: At the beginning of the course, many students were suspicious of the use of user stories for requirements specifications. They found user stories not precise enough to adequately describe the desired functionality. Therefore, the teacher should carefully communicate the advantages of user stories to instill the awareness that they serve mainly as reminders for conversation, while all details should be fleshed out in close communication with the Product Owner. Students need some time and practical experience to overcome their initial reluctance and fully grasp the concepts and advantages of user stories. In order to speed up the learning process, the teacher (in the role of ScrumMaster) should establish an environment that encourages regular communication between the Team and the Product Owner. Without a responsive Product Owner, the teaching of user story concepts is impossible.

6) Daily Scrum Meetings Must Not Be for Reporting to the ScrumMaster, but for the Team Members to Inform Each Other About the Current State of the Project: While some teams found the meetings useful (one student even reported that he had introduced such meetings in the company where he worked), it often happened that students ended up giving only a brief status report to the ScrumMaster instead of talking to each other about the project. Given that the teacher played the role of ScrumMaster, this behavior was quite understandable, but every effort must be made to ensure that the Daily Scrum meetings serve their primary purpose: to inform other team members about issues relevant to the successful implementation of their tasks, thus improving internal communication and teamwork.

7) Estimating and Planning Is a Difficult Task, but Can Improve Through Practice: Empirical data collected during the course indicate that the ability to plan improves from Sprint to Sprint. Most teams (even beginners) were able to define nearly accurate Sprint plans after three Sprints. In the third Sprint, the velocity stabilized, and the actual achievement almost completely matched that planned. Scrum contributes to improvement of students' estimation and planning skills by requiring them to reestimate their velocity and the remaining user stories at the beginning of each Sprint in order to obtain a more realistic plan for future iterations.

8) Conducting the Course as an Observational Study Is Beneficial for All Parties Involved: By monitoring the study, the teacher obtains much better feedback about what the students have learned. Empirical results can help him to analyze students' perceptions and the learnability of agile methods, find better ways of imparting the most important concepts, and identify areas that deserve more attention or improvement. Teachers who plan to teach a similar course can benefit from well-established empirical evidence. A well-designed study can provide students not only with a realistic simulation of professional experience, but can also introduce empirical software engineering as a part of software engineering teaching. Students can learn the importance of evidence-based assessment of agile methods in order to identify their strengths and weaknesses before actually deploying them in industrial software environments. By carefully considering the differences between the classroom and industry, the results of the study can also help industry by providing preliminary evidence of the behavior of development teams using Scrum for the first time—a typical situation for software companies trying to introduce Scrum into their development process.

9) Traditional Disciplined Approach Can Be Incorporated, but in an Appropriate Way That Preserves Agility: While students were overwhelmingly positive about the agile approach, they sometimes missed a more detailed up-front design that would give them a feeling of safety and of having taken the right direction toward the final outcome. A reasonable solution for this problem is to allow (or even encourage) students to use up-front design (as well as other elements of the disciplined approach to software development) within individual Sprints. Once the content of a Sprint is agreed with the Product Owner, the requirements for that Sprint are fixed; therefore, the use of a disciplined approach within a Sprint does not hinder the agility of the development process as a whole. On the contrary, combining the best practices of both approaches can lead to a stable design that evolves stepwise from Sprint to Sprint. The final decision is up to each team. Bearing in mind the Scrum principles of self-organizing and self-managing teams, the Team is responsible for figuring out the best way to implement the required functionality.

ACKNOWLEDGMENT

The author thanks teaching assistants L. Fürst and T. Hovelja for their help in conducting the course and analyzing data on student teams' performance.

REFERENCES

[1] 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 [Online]. Available: http://sites.computer.org/

[2] T. C. Lethbridge, R. J. LeBlanc, Jr., A. E. Kelley Sobel, and J. L. Díaz-Herrera, "SE2004: Recommendations for undergraduate software engineering curricula," *IEEE Softw.*, vol. 23, no. 6, pp. 19–25, Nov.–Dec. 2006.

[3] S. Tilley, S. Huang, K. Wong, and S. Smith, "Report from the 2nd International Workshop on Software Engineering Course Projects (SWECP 2005)," in *Proc. 19th Conf. Softw. Eng. Educ. Train.*, Turtle Bay, HI, 2006, pp. 87–94.

[4] S. Smith, M. Mannion, and C. Hastie, "Encouraging the development of transferable skills through effective group project work," in *Software Engineering in Higher Education II*, J-L. Uso, P. Mitic, and L. J. Sucharov, Eds. Southampton, U.K.: Computational Mechanics, 1996, pp. 19–26.

[5] A. Mohan, D. Merle, C. Jackson, J. Lannin, and S. S. Nair, "Professional skills in the engineering curriculum," *IEEE Trans. Educ.*, vol. 53, no. 4, pp. 562–571, Nov. 2010.

[6] S. Karunasekera and K. Bedse, "Preparing software engineering graduates for an industry career," in *Proc. 20th Conf. Softw. Eng. Educ. Train.*, Dublin, Ireland, 2007, pp. 97–106.

[7] P. Abrahamsson, O. Salo, J. Ronkainen, and J. Warsta, *Agile Software Development Methods*. Espoo, Finland: VTT Electronic, 2002.

[8] D. Cohen, M. Lindvall, and P. Costa, "An introduction to agile methods," *Adv. Comput.*, vol. 62, pp. 2–67, 2004.

[9] 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," *Int. J. Eng. Educ.*, vol. 24, no. 4, pp. 659–670, 2008.

[10] D. F. Rico and H. H. Sayani, "Use of agile methods in software engineering education," in *Proc. Agile 2009 Conf.*, Chicago, IL, 2009, pp. 174–179.

[11] D. A. Umphress, T. D. Hendrix, and J. H. Cross, "Software process in the classroom: The capstone project experience," *IEEE Softw.*, vol. 19, no. 5, pp. 78–85, Sep.–Oct. 2002.

[12] C. Coupal and K. Boechler, "Introducing agile into a software development capstone project," in *Proc. Agile Develop. Conf*, Denver, CO, 2005, pp. 289–297.

[13] S. Ramakrishnan, "Innovation and scaling up agile software engineering projects," *Issues Informing Sci. Inf. Technol.*, vol. 6, pp. 557–575, 2009.
[14] P. A. Laplante, "An agile, graduate, software studio course," *IEEE Trans. Educ.*, vol. 49, no. 4, pp. 417–419, Nov. 2006.
[15] V. Devedzic and S. R. Milenkovic, "Teaching agile software development: A case study," *IEEE Trans. Educ.*, vol. 54, no. 2, pp. 273–278, May 2011.
[16] P. N. Robillard and M. Dulipovici, "Teaching agile versus disciplined processes," *Int. J. Eng. Educ.*, vol. 24, no. 4, pp. 671–680, 2008.
[17] T. Dingsøyr, T. Dybå, and P. Abrahamsson, "A preliminary roadmap for empirical research on agile software development," in *Proc. Agile 2008 Conf.*, Toronto, ON, Canada, 2008, pp. 83–94.
[18] T. Dybå and T. Dingsøyr, "Empirical studies of agile software development: A systematic review," *Inf. Softw. Technol.*, vol. 50, no. 9-10, pp. 833–859, Aug. 2008.
[19] "4th annual survey: The state of agile development," VersionOne, Atlanta, GA, 2009 [Online]. Available: http://www.versionone.com/pdf/2009_State_of_Agile_Development_Survey_Results.pdf
[20] K. Schwaber and M. Beedle, *Agile Software Development with Scrum*. Upper Saddle River, NJ: Prentice-Hall, 2002.
[21] K. Schwaber, *Agile Project Management with Scrum*. Redmond, WA: Microsoft Press, 2004.
[22] V. Mahnic, "Teaching Scrum through team-project work: students' perceptions and teacher's observations," *Int. J. Eng. Educ.*, vol. 26, no. 1, pp. 96–110, 2010.
[23] J. Carver, L. Jaccheri, S. Morasca, and F. Shull, "Issues in using students in empirical studies in software engineering education," in *Proc. 9th Int. Softw. Metrics Symp.*, Sydney, Australia, 2005, pp. 239–249.
[24] J. C. Carver, L. Jaccheri, S. Morasca, and F. Shull, "A checklist for integrating student empirical studies with research and teaching goals," *Empirical Softw. Eng.*, vol. 15, pp. 35–59, 2010.
[25] M. Cohn, *User Stories Applied for Agile Software Development*. Boston, MA: Addison-Wesley, 2004.
[26] J. Grenning, "Planning poker or how to avoid analysis paralysis while release planning," Apr. 2002 [Online]. Available: http://renaissancesoftware.net/files/articles/PlanningPoker-v1.1.pdf
[27] "Agilo—Flexible software for Scrum management," agile42, North Vancouver, BC, Canada, 2010 [Online]. Available: http://www.agile42.com/cms/pages/agilo/
[28] V. Mahnic, I. Rozanc, and M. Pozenel, "Using E-business technology in a student records information system," in *Proc. 7th WSEAS Int. Conf. E-Activities*, Cairo, Egypt, 2008, pp. 80–85.
[29] L. Baresi and S. Morasca, "Three empirical studies on estimating the design effort of Web applications," *Trans. Softw. Eng. Method.*, vol. 16, no. 4, Sep. 2007, Article 15.
[30] M. Jørgensen, "A review of studies on expert estimation of software development effort," *J. Syst. Softw.*, vol. 70, no. 1-2, pp. 37–60, Feb. 2004.
[31] N. E. Fenton and S. L. Pfleger, *Software Metrics*. Boston, MA: PWS, 1997.

Viljan Mahnic (M'00) received the B.S., M.S., and Ph.D. degrees in computer science from the University of Ljubljana, Ljubljana, Slovenia, in 1978, 1981, and 1990, respectively.

He is an Associate Professor of computer science and Head of the Software Engineering Laboratory, Faculty of Computer and Information Science, University of Ljubljana. His teaching and research interests include agile software development methods, software process improvement, empirical software engineering, and software measurement.