Teaching Agile the Agile Way — Employing Self-Organizing Teams in a University Software Engineering Course

Introduction

Teaching agile software methodologies in university courses has become the norm, as these methods have gained support in professional software development. There are different approaches of how to teach these methods in university courses. Traditional lectures and seminars are proven, effective tools for teaching factual knowledge and technical skills. However, as Bruegge et al.[5] point out, in order to "really assimilate and internalize the agile values, a theoretical lecture is not enough". In consequence, educators have included practical projects in their software development course designs, recognizing the need to learn about the human (i.e. cognitive and social) aspects of software engineering.[9] These practical projects are able to complement lectures and teach a different set of skills. Kropp and Meier[11] introduced the Agile Competence Pyramid model, differentiating various levels of these agile skills and rating them in importance. Agile values are considered to be most important in their model by being on the top of the pyramid. However, the authors note that this level of the model "is mostly completely neglected" in current course programs. In order to address this issue in an integrated approach, we have developed the *software engineering II* course for undergraduates of the Hasso-Plattner-Institute, Potsdam. It combines an extensive, hands-on student project, intensive coaching as well as a minimal amount of theoretical lectures on software development methodologies and best practices. One of the core aspects of the course is the focus on communicating the agile values, outlined in the Agile Manifesto,[4] such as "individuals and interactions over processes and tools" through giving students authority and responsibility for decisions and role assignment and letting them self-organize. Self-organizing teams are one of the core concepts of agile development methodologies[6] and have been identified as critical to the success of an agile project.[3]
In this paper, we present a course design focusing on empowering students to discover the benefits of agile practices autonomously, encouraging them to take ownership of the process. Furthermore, we describe the surveys, that were conducted and compare their results to those of alternative course designs.

Related Work

There are many recent accounts in the literature of courses teaching Scrum in a project-oriented fashion. Mahnic[12–14] provides a detailed account of the software engineering course at Ljubljana University. The paper contains details about the surveys that were conducted at the end of sprints and the results thereof and parts of the surveys we conducted are based on the ones presented there. Mahnic used the *Earned Value Method* (EVM)[2] project management technique and its associated performance indexes in addition to Scrum burndown charts as tools to gain insight into project performance. This came at the cost of additional documentation workload for students.

Devedzic et al.[7] summarize their experience gained through teaching university software engineering courses. In contrast to Mahnic, they urge that teaching itself should follow "the agile way", e.g. introducing stand-up meetings at the beginning of classes. The authors furthermore present lessons learned including assigning the Scrum Master role to students and favoring practice over theoretical discussion. The authors collected quantitative data in order to ascertain the students' productivity, including Java classes written and performed refactorings.
As there are currently no unified standards for teaching agile methodologies[1] every course instructor needs to find a setup, that works within their context and with the knowledge that students have already acquired. In our course setup we have integrated ideas from the presented approaches, while opting to focus on agile values and their application in the course itself.

Course Setup

The course presented in this paper is a final year undergraduate software engineering course, in which multiple self-organizing teams of students jointly develop a single software system, while employing the Scrum methodology. The goal is to provide students with a project experience that provides real challenges in the context of agile software development. Students are consciously allowed to make mistakes and learn from them.
The selection of the software to be developed is based on the actual needs of university staff. In the 2014/15 installment, a system for the event and room planning of the university was created. All participants previously attended lectures teaching the fundamentals of software engineering, including core principles of agile development processes. However, the presented course is the first one in the Bachelor curriculum that applies this knowledge to collaborating with more than 30 developers in mid-sized development teams.
Prior to the beginning of the course, an online introduction exercise[1] aims to familiarize participants with the Ruby on Rails web development framework that is being employed. To refresh Scrum knowledge, a Lego Scrum simulation exercise[16] is performed in the second week of the course, prior to the start of the project. It is supervised by the teaching team and plays through multiple iterations in a short timespan, giving an overview of the process, its roles and practices. Furthermore, short lectures about core elements of agile development, such as Test-Driven Development, Behavior-Driven Development, Continuous Integration and Automated Deployment are given throughout the semester.
The course is allocated 6 ECTS (European Credit Transfer and Accumulation System) credit points,[8] resulting in a total expected workload per student of about 8 hours per week. Two product presentations are held and organised by students: an intermediate one halfway through the course, and a final one at the end of the course. The Product Owners, with help by team members, present the system to the entire course as well as the teaching team and other stakeholders, i.e. the future users of the system, and receive feedback.
Within the project, a custom-tailored version of the Scrum process is used as a framework for development efforts (see Figure 1). The teams follow the Scrum process as closely as possible, with necessary adaptations being made to account for the fact that students are not full-time developers. XP (eXtreme Programming) practices are also taught, as their

[1] https://www.codeschool.com/courses/rails-for-zombies-redux

focus on engineering complements the management practices of Scrum and we agree that "Scrum and XP can be fruitfully combined".[10]

Based on the number of attendees, a varying number of teams are formed by the students themselves. Each team typically consists of 5 to 8 members. The teams jointly select one Product Owner (PO) and a Scrum Master (SM) from among them, the remainder of the students act as developers. We encourage students to select the person with the greatest knowledge of the employed programming languages as SM, so that she is able to effectively coach other team members. POs and SMs, due to the inherent complexity of their roles are given additional attention. The POs are offered the assistance of a "Head PO", an experienced member of the teaching team, who is available for meetings and provides support in the domain of user story creation, formalization and prioritization. SMs receive additional feedback by tutors at the end of meetings on how to manage meetings, keep time and ensure the team works productively and adheres to agile practices.

The course takes place in the winter term and allows for 12 weeks of active development, which are split into 4 sprints of three weeks duration. Within each week, students are required to spend 8 hours on the course, including participation in lectures. For each sprint, a planning meeting, weekly Scrum meetings, a sprint review, and a retrospective meeting are organized by the students and performed in the presence of a tutor. The teams schedule these meetings autonomously and find optimal time slots that fit all participants. Additional meetings, such as Scrum-of-Scrum meetings or gatherings of the POs are optional. In recent installments of the course, tutors have proposed these in order to help with communication problems. As student working time is supposed to be limited to 8 hours per week, daily standup meetings are replaced by weekly versions.

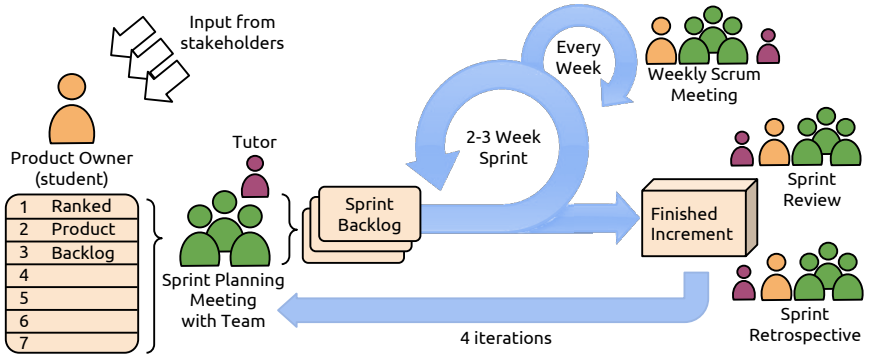

Fig. 1. The modified Scrum process followed in the course software engineering II.

Two weeks prior to the project's start, the POs meet with the customer and discuss initial requirements for the system. The customer is a member of the teaching team who has intricate knowledge of the problem domain and represents the gathered requirements of stakeholders. Under guidance by the teaching staff, the teams themselves consequently fill the product backlog and decide which team is responsible for the implementation of which aspects of the system. Once these preparations are finished, the POs present the overall product vision and responsibilities to the other students. Subsequently, each team performs a sprint planning meeting to decide on the sprint backlog and identify potential dependencies to other teams. After two to three weeks of development, a sprint review and retrospection concludes the sprint. During sprints, the PO team presents intermediate progress to the customer in order to clarify existing and elicit new requirements.

When giving teams authority to self-organize and adapt their process and meetings according to the team's needs, intensive coaching and feedback is vital to allow students to learn from their mistakes. This is facilitated through tutors, research assistants with an understanding of Scrum and its best practices. Tutors need to be present during Scrum meetings to answer questions, give advice when problems occur. In our experience, the tutor role works best if understood as a consultant working with the team rather than a teacher. The tutor can encourage the Scrum Master to actively lead meetings and give live feedback to the Product Owner on user stories and prioritization.

Surveys

The surveys were conducted with all participants of the course. They assessed students' satisfaction with Scrum and its associated practices over the span of the course. The Survey data was analyzed regarding its correlations, giving insights into what areas of the Scrum process are related and what key components should be measured and improved to affect larger parts of processes.
The test subjects were 38 students, 6 females and 32 males: students in their final two semesters of undergraduate studies. For each participant who filled out the survey, the corresponding team was identified.
Two distinct surveys were employed:

1. **Sprint Survey**. Survey tracking key aspects of agile practices. These surveys were conducted at the end of each sprint, directly after the team's sprint retrospective, before the sprint planning meeting of the next iteration.
2. **Feedback Survey**. Survey tracking student's perceptions of the course in retrospective, measuring satisfaction with the overall course and its design.

For all surveys subjects were completely anonymous within their teams and were not tracked over time. We felt this was necessary to preserve the students' trust in the grading process of the course.

Sprint Survey (S1)

In order to allow comparisons to other courses, an existing questionnaire by Mahnic,[12] used in their final undergraduate software engineering course in 2008, was adapted. Mahnic employed a more controlled course setup, allowing comparisons between course design approaches. Like the original survey, the sprint survey consisted of 10 questions, answerable on a 5-point Likert scale: "strong no" (1) to "strong yes" (5), with 3 being "neutral". Some questions of the original survey had to be adapted to fit our context. References to requirements were replaced by references to user stories and references to the specifics of Mahnic's course, i.e. a "spreadsheet application" used track data, were removed. The adapted survey, that was used in the course, is given in Table 1.
The results of S1 are presented in Table 2. For each question, the columns Sprint 1 through 4 show the calculated mean rating, the standard deviation as well as the one-sample t-test of the mean for each sprint. The last column shows the results of an analysis of variance (ANOVA) on the team means for one factor (sprint) as a within-subjects design.
The null hypothesis of students having a neutral attitude towards Scrum, was rejected in 32 of the 40 question means. In all these cases, we found positive average ratings. This indicating a positive attitude towards Scrum and its practices. We can therefore lend some

#	Question
1	**Clarity of requirements in the Product Backlog** Were the user stories in the backlog clear enough? Did the descriptions suffice to understand what the Product Owner really wanted?
2	**Effort estimation** Were the estimates of required work (story points / man-hours) of user stories adequate / realistic?
3	**Maintenance of the Sprint Backlog** Was it clear how to handle user stories? How to log performed work?
4	**Administrative workload** Does the administrative work called for by Scrum (meetings, planning, reviews, maintenance of backlog, etc.) represent a significant additional workload?
5	**Cooperation with the Scrum Master** Was the cooperation with the Scrum Master adequate / satisfactory? Did the Scrum Master contribute to the team's success?
6	**Cooperation with the Product Owner** Was the cooperation with the Product Owner adequate / satisfactory?
7	**Cooperation with other Team members** Was the cooperation with other Team members satisfactory / adequate? Did the Scrum process encourage cooperation?
8	**Workload** Was the amount of work required for the project adequate?
9	**Satisfaction with work** Are you satisfied with the work / the results of this sprint?
10	**Satisfaction with Scrum** Is the methodology useful? Would you recommend Scrum to other software developers?

Table 1. Sprint survey. Questions could be answered on a 5-point scale of "strong no" to "strong yes".

credibility to the hypothesis that students on average had a positive attitude towards the use of the Scrum methodology in our course featuring self-organizing teams. In particular, the average rating for question 10, asking whether students were satisfied with Scrum and would recommend it, only sank below 4 ("yes") once, after Sprint 2 (average rating of 3.82).
In order to identify correlations between responses to questions, the Pearson product-moment correlation coefficient was calculated for all questions in the sprint survey. Of the 45 possible correlations, 27 became significant at the <0.05 level. Table 3 shows the correlations between answers to S1.
Significant correlations were found between answers to all of the questions dealing with collaboration of team members and Scrum roles (q5, q6, q7). Furthermore, Satisfaction with the Scrum method (q10) showed significant positive correlations with q5, q6 and q7. This highlights the importance of a well-functioning team. The hypothesis drawn from this is that practicing good collaboration with parts of a self-organizing team will favorably influence the outlook on the other team members and the team atmosphere. Problems in collaboration with parts of the team might spread to collaboration in other areas.

Comparison with other Course Setups The 2014/15 installment of the software engineering course analyzed here can be compared with the 2008/09 course by Mahnic,[12] as the same survey was used. Both courses received consistently positive scores over all sprints for the last two questions of the survey measuring satisfaction with the performed work and the Scrum methodology. These findings regarding student perceptions of agile methods are in line with other, more extensive studies.[15]

	Sprint 1			Sprint 2			Sprint 3			Sprint 4			
#	Mean	Std. dev.	One-sample t-test (t-value)	Mean	Std. dev.	One-sample t-test (t-value)	Mean	Std. dev.	One-sample t-test (t-value)	Mean	Std. dev.	One-sample t-test (t-value)	ANOVA over all sprints (f-value)
1	3.56	0.84	3.79***	3.81	0.83	5.39***	3.97	0.51	11.38***	4.00	0.57	9.96***	1.47
2	2.97	0.93	-0.19	2.76	1.13	-1.19	3.16	0.81	1.16	3.61	0.86	4.03***	1.21
3	3.28	1.05	1.51	3.74	1.12	3.67***	3.91	0.73	7.43***	4.13	0.75	8.47***	6.52**
4[1]	3.50	0.95	2.98**	3.10	1.14	0.47	3.06	0.82	0.42	3.20	0.93	1.27	1.49
5	4.18	0.73	9.34***	4.22	0.75	9.18***	4.26	0.60	12.26***	4.33	0.69	11.07***	0.43
6	3.78	0.83	5.31***	4.13	0.75	8.47***	4.04	0.81	7.68***	4.26	0.73	9.61**	1.22
7	3.63	0.91	4.09***	3.91	1.01	5.16***	4.01	0.54	11.25***	4.07	0.64	9.85***	2.44"
8	3.68	0.67	6.14***	3.26	0.96	1.60	3.42	0.64	4.04***	3.49	0.84	3.47**	1.53
9	3.41	0.90	2.75**	3.40	1.12	2.12*	3.40	0.89	2.74**	3.94	0.86	6.59***	1.78
10	4.14	0.67	10.25***	3.82	0.87	5.52***	4.07	0.47	13.99***	4.10	0.61	10.84***	1.44

Table 2. Data collected in S1, answers over the whole body of students. ***<0.001, **<0.01, *<0.05, "<0.15, [1] indicates items with inverse item coding.

Survey question	1.	2.	3.	4.	5.	6.	7.	8.	9.
1. BLclarity									
2. Estimation	0.04								
3. BLmaint	0.14	0.14							
4. AdminWork	-0.31***	0.18*	-0.18*						
5. SMcollab	0.00	0.12	0.49***	-0.12					
6. POcollab	0.49***	0.08	0.31***	-0.23**	0.28***				
7. TeamCollab	0.34***	-0.10	0.17*	-0.21*	0.16*	0.32***			
8. Workload	0.23**	0.22**	0.14	-0.11	0.28***	0.24**	0.11		
9. SatisfiedWork	0.30***	0.32***	0.14	-0.03	0.15	0.39***	0.30***	0.27***	
10. SatisfiedScrum	0.06	0.09	0.26**	-0.11	0.24**	0.23**	0.20*	0.09	0.19*

Table 3. Correlations between answers to S1. The topics of questions are abbreviated. Correlation significance: ***<0.001, **<0.01, *<0.05.

While the course described by Mahnic lasted only two sprints, we observed students for four sprints. Furthermore, Mahnic's course setup mainly differed from ours in the level of control exerted over students.

In Mahnic's course setup the roles of PO and SM were assigned to a member of the teaching team. In our setup they were assigned to students. However, scores for these aspects were similar. The cooperation with the SM (q5) consistently received positive average scores (over 4) in both courses. Similarly, collaboration with the PO (q6) was consistently rated positive in both course setups. In fact, collaboration with the SM (q5) was rated the highest out of all questions in all sprints in both courses. In Mahnic's course, students were supplied with a prefilled product backlog that only needed estimations, while POs in our course were tasked with creating all needed user stories themselves from scratch. However, the average grade of questions concerning the maintenance (q3) and clarity of the backlog (q1) were consistently positive over the entirety of the course in both course setups.

This lends evidence to the hypothesis that assigning Scrum roles to students, instead of teaching team members, does not have a negative impact on collaboration in teams. Mahnic reports that no significant correlations between the satisfaction with Scrum (q10) and other questions could be found. He states, however, that it "students' satisfaction with Scrum depends mostly on the ease of the Sprint Backlog maintenance (r=0.33) and the amount of additional administrative workload (r=0.24)".[12] In our setup, a significant correlation with backlog maintenance (q3, r=0.26) was present. Furthermore, additional positive correlations with Scrum satisfaction, absent from Mahnic's results were found (see Table 3), including collaboration with the SM (q5), PO (q6) and the team (q7) as well as with satisfaction with the performed work (q9). These additional correlations concern questions about Scrum roles, which were assigned to students instead of members of the teaching team, as Mahnic did. This adds to the hypothesis that students in Scrum roles do not negatively impact the collaboration of teams.

Feedback Survey (S2)

In addition to S1, a more general feedback survey, was conducted at the end of the last sprint in order to measure students' perceptions of the course in retrospective. S2 gathered feedback about the specific design choices made in the course setup. Like in S1, questions could be answered on a 5-point Likert scale. All 38 students participated in the feedback survey. Table 4 lists the questions as well as the results of the mean, standard deviation and one-sample t-test for each one.

#	Topic	Mean	Std. dev.	One-sample t-test (t-value)
1	Are you confident in applying the Scrum process?	4.29	0.46	16.31***
2	Do you feel you acquired meaningful skills for the future?	4.67	0.53	18.28***
3	Was it sensible to run the course as a large programming project?	4.37	0.71	11***
4	Should there have been additional lectures? If so, which ones?	3	1.14	0
5	Was it helpful that participants had to mostly self-organize?	3.44	1.03	2.5202*

Table 4. Results of S2. Significance: *** < 0.001, **<0.01, *<0.05.

On average, students answered that they had acquired meaningful skills for the future (mean score of 4.67). Students felt confident in applying the Scrum methodology (q1) and agreed that running the course as a large programming project was sensible (q3), both reaching mean scores over 4 on the 5-point scale.
Questions regarding additional lectures (q4) and letting teams self-organize (q5) received average scores of 3 and 3.44 respectively. Answers to both questions feature standard deviations of more than one point. This disparity was also evident in verbal feedback. Some students appreciated few lectures and a focus on self-organization, while others preferred more tutorials and structure. Reconciling these two sides in a software engineering course is an ongoing challenge. Even so, the focus of the course on self-organizing teams did not hinder the perceived learning results of students.

Conclusion

Both surveys that were conducted throughout the course showed positive student attitudes towards the self-organizing approach. Furthermore, results did not significantly differ from

those of more controlled course setups. However, one of the consequences of encouraging student teams to self-organize is the decrease in direct control over the executed processes in teams. To compensate for this, intensive coaching of teams is necessary, making sure that tutors with extensive knowledge of agile methods and their application are available.[18] These tutors answer questions, guide students and can make suggestions if they notice that a team has not self-corrected a problem in the adoption of agile practices after some time. They make sure that the executed process of a team remains agile and that core agile practices are adhered to correctly.[17] While this approach requires a definite increase in the amount of time spent on supervision and support of teams, it makes up for it by allowing students to discover the benefits of agile values for themselves. By being allowed to make mistakes and leveraging agile practices to overcome them, students are able to discover and internalize the benefits of agile values.

References

1. ACM/IEEE-CS Joint Task Force on Computing Curricula. Computer science curricula 2013. Technical report, ACM Press and IEEE Computer Society Press, December 2013.
2. F. T. Anbari. Earned Value Project Management Method and Extensions. *Project Management Journal*, 34(4):12–23, 2003.
3. S. Augustine. *Managing agile projects*. Prentice Hall PTR, 2005.
4. K. Beck, M. Beedle, A. Van Bennekum, W. Cockburn, Alistair Cunningham, M. Fowler, J. Grenning, J. Highsmith, A. Hunt, and R. Jeffries. Agile Manifesto, 2001.
5. B. Bruegge, M. Reiss, and J. Schiller. Agile principles in academic education: A case study. *ITNG 2009 - 6th International Conference on Information Technology: New Generations*, pages 1684–1686, 2009.
6. T. Chow and D.-B. Cao. A survey study of critical success factors in agile software projects. *Journal of Systems and Software*, 81:961–971, 2008.
7. V. Devedzic and S. R. Milenkovic. Teaching Agile Software Development: A Case Study. *IEEE Transactions on Education*, 54:273–278, 2011.
8. European Commission. *ECTS Users' Guide: European Credit Transfer and Accumulation System and the Diploma Supplement*. European Commission, 2005.
9. O. Hazzan and Y. Dubinsky. Why Software Engineering Programs Should Teach Agile Software Development. *Software Engineering Notes*, 32(2):1–3, 2007.
10. H. Kniberg. *Scrum and XP from the Trenches*. C4Media, 2007.
11. M. Kropp and A. Meier. Teaching agile software development at university level: Values, management, and craftsmanship. In *Software Engineering Education Conference, Proceedings*, pages 179–188, 2013.
12. V. Mahnič. Teaching Scrum through Team-Project Work : Students' Perceptions and TeacherâĂŹs Observations. *International Journal of Engineering Education*, 26:96–110, 2010.
13. V. Mahnič. A Capstone Course on Agile Software Development Using Scrum. *IEEE Transactions on Education*, 55(1):99–106, 2012.
14. V. Mahnič, S. Georgiev, and T. Jarc. Teaching Scrum in Cooperation with a Software Development Company. *Organizacija*, 43(1):40–48, 2010.
15. G. Melnik and F. Maurer. A cross-program investigation of students' perceptions of agile methods. In *Proceedings of the 27th International Conference on Software Engineering*, pages 481–488, may 2005.
16. M. Paasivaara, V. Heikkilä, C. Lassenius, and T. Toivola. Teaching Students Scrum using LEGO Blocks. *Companion Proceedings of the 36th International Conference on Software Engineering - ICSE Companion 2014*, pages 382–391, 2014.
17. A. Ruhnow. Consciously evolving an agile team. *Proceedings - AGILE 2007*, pages 130–135, 2007.
18. C. J. Stettina, Z. Zhao, T. Back, and B. Katzy. Academic education of software engineering practices: Towards planning and improving capstone courses based upon intensive coaching and team routines. *Software Engineering Education Conference, Proceedings*, pages 169–178, 2013.