Teaching Agile Software Development at University Level[1]

Several recent surveys show that agile methodologies like Scrum, Extreme Programming and Kanban have been successfully adopted by many companies for software development. However, the same surveys show that only few of the agile practices are applied consequently and thoroughly. This is to a great extent due to the lack of skilled personnel. In this paper we propose a more holistic approach for teaching agile software development, in which the required agile practices and values are not only integrated theoretically into our courses but also practically applied. The proposed concept was realized in a new a course at Zurich University of Applied Sciences during 2012. The evaluation shows very encouraging results.

Martin Kropp, Andreas Meier | martin.kropp@fhnw.ch

Recent surveys [1,2] show that agile methodologies in many respects deliver better outcomes than plan-driven ones. As a result, agile software development has been adopted by many IT companies and IT departments. In the Swiss Agile Study (SAS), a survey conducted by the authors, these findings have been confirmed [2]. More than half of the participating companies are using an agile methodology like Scrum [14] or XP [15] – Agile has become mainstream!

Unfortunately, this also has a significant impact on the agile team constitution. The early adopters of agile approaches were all highly mature and technically skilled experts in their fields. They had internalized the agile philosophy, were very productive and produced high quality results. Today's agile teams, however, are "normal" software teams, with architects, seniors and juniors in one team, and many of them are not yet familiar with the agile philosophy. Even though those teams have improved in software development to some extent, they are far less productive than the early adopter expert teams. Survey results show that quality has partially even gone down and overall costs increased. One reason for this may be that many of the important agile practices are not applied as thoroughly as the agile pioneers proposed [13].

In this paper we will analyze the situation on the industry side in more detail to find out, which skills are missing and make a proposal how education on university level can help to improve this. We will suggest a holistic teaching approach, which integrates the necessary agile engineering and managements skills together with the core agile values, into the education of agile software development.

In the next two sections we give an overview of related work to set our paper in context and we analyze the reasons for the rather poor performance of today's agile teams and contrast this with the current state of software development education. Then we present the Agile Competence Pyramid as a model for the required competences for agile software development. In the rest of the article we present the layout and the evaluation of a new Software Engineering course, which was held at the Zurich University of Applied Sciences in the 5th semester of the undergraduate Computer Science program. We conclude with an outlook on further work.

Related Work

Though agile software development has been around for more than a decade (even before the famous Agile Manifesto [13]), teaching agile software development has only drawn some attention in educational and research conferences in the last few years. A reason for this might be that agile development is not based on a green-field theory but has been developed from practice. In [3] the authors discuss reasons why software engineering programs should teach agile software development. They emphasize that software engineers not only need technical skills but also social and ethical ones, which are both corner stones of agile development. In [4] the authors emphasize that theoretical lectures about agile development are not enough, but that students have to apply agile methods to really internalize them. The authors present a case study with 80 students working on a large project. There are several recent case study papers and experience reports [5-8] in which the authors report about their experiences teaching agile software engineering courses.

Motivation

The recent Swiss Agile Study, in which 140 Swiss IT companies and almost 200 IT professionals participated, shows very clear results. IT com-

1 The original version of this paper has been published at the 26th Conference on Software Engineering Education and Training, May 19–21, 2013. CSEE&T '13, San Francisco, USA.

Aspect	much worse	worse	unchanged	improved	significantly improved	don't know
Changing priorities	1%	0%	9%	45%	44%	1%
Development process	0%	2%	17%	58%	22%	1%
Time to market	1%	2%	19%	53%	23%	2%
Alignm.btw. IT and business	0%	1%	25%	46%	23%	6%
Project visibility	0%	2%	25%	39%	28%	6%
Team morale	0%	4%	25%	42%	24%	5%
Requirements management	0%	2%	29%	51%	13%	5%
Productivity	0%	2%	33%	47%	15%	4%
Risk management	0%	5%	32%	42%	17%	4%
Software quality	0%	2%	45%	35%	16%	2%
Software maintainability	0%	7%	55%	23%	12%	3%
Development cost	1%	12%	52%	22%	7%	6%
Engineering discipline	0%	4%	42%	42%	9%	4%

Table 1: How has agile software development influenced the following aspects?

panies and IT professionals following the agile methods are much more satisfied with their methodologies than their plan-driven counterparts. The study also shows very clearly, that major goals of introducing agile development have been reached: A significant improvement in the ability to manage changing priorities, improvement of the development process in general and a much faster time-to-market.

Table 1 summarizes the influence of agile software development as given by the participating agile IT companies. Though the survey shows very promising results at first view, there are also quite astonishing findings. It is reported that development cost, software quality and software maintainability have not improved as much as expected. With respect to development cost and software maintainability, 7%, respectively 12% of the participants reported that these have even got worse. This clearly contradicts the intention of the authors of the agile manifesto, who want to deliver high quality code that is easily maintainable.

One reason for this might lie in the fact that only few of the agile practices are used consistently throughout the whole software development process. While engineering practices like coding

Items	completely disagree	disagree	agree	completely agree
Computer Science graduates (M.Sc.) have sufficient knowledge of agile methodologies	5%	53%	33%	9%
Computer Science undergraduates (B.Sc.) have sufficient knowledge of agile methodologies	8%	60%	28%	4%

Table 2: Knowledge of graduates

Items	completely disagree	disagree	agree	completely agree
Agile development should be an integral part of the Computer Science curriculum	0%	5%	49%	46%
Agile should not be taught at university, it is better learned on the job	34%	48%	12%	7%

Table 3: Agile as part the computer science curriculum

standards, unit testing or automated builds are used by two-third or more of the agile companies, other necessary practices like continuous integration, refactoring, test-driven development are used by only half the participants or even less. A similar result is obtained with respect to the management practices: while two-third or more of the participants use iteration planning, release planning or user stories, only half or even less of the participants use daily standups, task boards or retrospectives.

The SAS shows, that there are too few software engineers with the skills for agile development. This suggests that we as teachers do not yet educate the students with the required skills. This assumption is backed by answers in Table 2. Almost 70% percent of the participating companies think that undergraduates have too little knowledge of agile; still the majority thinks this is true for graduates.

Table 3 answers the questions, whether agile development should be an integral part of the computer science curriculum. The majority of the participants recommend that agile software development should be an integral part of the computer science curriculum. As educators, we have to take the findings from the above tables seriously and try to make sure that future graduates will have sufficient knowledge of agile methodologies.

Evaluation of Effort and Learning Effect

Two major characteristics of agile software development are its focus on working software over

		far too high	too high	exactly right	too little	far too little
Lecture	Documentation	4%	25%	36%	23%	12%
	Management	2%	16%	45%	27%	11%
	Programming	4%	9%	50%	27%	10%
Project	Documentation	32%	38%	27%	3%	1%
	Management	18%	31%	40%	10%	1%
	Programming	2%	8%	40%	34%	16%

Table 4: How do you estimate the effort for the different activities?

		very high	high	low	very low
Lecture	Documentation	0%	22%	47%	31%
	Management	2%	29%	51%	18%
	Programming	18%	45%	24%	14%
Project	Documentation	8%	29%	51%	12%
	Management	10%	39%	38%	13%
	Programming	18%	42%	28%	12%

Table 5: How do you estimate the learning effect?

documentation and lightweight management. Therefore, the authors wanted to know how much effort computer science student spend on programming, management and documentation in the lectures and in their student projects, and about the learning effect they got from these activities. Table 4 and Table 5 show the results of an evaluation the authors conducted from 103 students at the two Universities of Applied Sciences in Zurich and Northwestern Switzerland.

Table 4 shows that the students estimated the effort spent for the different activities in the lectures more or less appropriate. In the student project, however, the majority of the students estimated the effort spent for documentation and management too high or even far too high.

Table 5 shows that the students estimated the learning effect of management activities significantly higher in student projects than in lectures. Interesting is the result for the documentation activity. The learning effect for software project documentation was seen to be much lower in the student project than in lectures. Setting this in relation to the results from Table 4 might suggest that the wrong style of documentation was taught in the student project.

The perceived results of this evaluation support the authors' hypothesis, that too much time is spent on agile management practices and, even worse, on documentation. The strong focus on documentation might come from the still existing influence of plan-driven methodologies.

Pyramid of Agile Competences

Before developing a new agile software engineering course, it is important to analyze the needed skills and competences for agile software development. The required competences can be divided into three major categories:

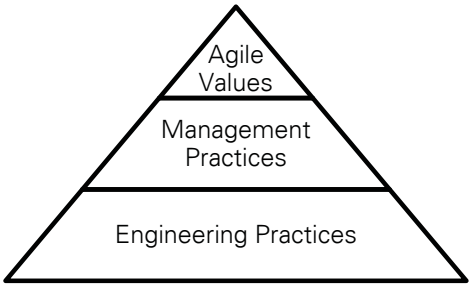

Figure 1: Pyramid of agile competences

Mastering the technical skills or engineering practices, builds the foundation for being able to develop high quality software. These engineering practices are especially defined by eXtreme Programming and include best practices like unit testing, clean coding, test-driven development, collective code ownership and the like. Engineering practices are mostly competences that refer to the single individual.

On the second level come the agile management practices. They define how agile projects are organized and run. Agile management practices include iterative planning, short release cycles, small releases, strong customer involvement and highly interactive teams. Management practices are typically team aspects, which require the appropriate social competences.

On top of these competences come the agile values, which are articulated in the agile manifesto and are based on characteristics like mutual respect, openness, and courage. Figure 1 visualizes the required competences in an Agile Competence Pyramid.

The pyramid visualizes the decreasing number of required skills from bottom to top. On the other hand, it reflects the increasing difficulty to teach these skills. Engineering practices can be taught very well in the classroom through lecturers and be learned by the individuals at their own pace. Management competences are best taught through student projects in teams, as our student evaluation confirms. The most difficult competences to teach are the values on top of the pyramid, since they often require a change in the attitude of the individual.

These different competence levels have to be considered in an agile software engineering course and have guided the authors in the design of the new course.

Agile Software Engineering Course

The course was a typical 16-week semester class in the last year of the undergraduate level (B.Sc.). The students completed one Java programming

project in an agile team of six to eight members during the course of the semester. Per week there were a two hours lecture with the whole class and a two hours programming workshop with half the class. 27 students were enrolled.

The scope of the course was equivalent to four ECTS credit points (European Credit Transfer and Accumulation System, one credit point is equivalent to 30 hours of studying). The course consisted of lectures (32 hours), workshops (32 hours), and self-study including programming (56 hours).

The authors have successfully used a Scrum-XP-hybrid for many years and therefore decided to use it in this course as well. Why do Scrum and XP work well together? Scrum focuses on management practices while XP focuses mostly on engineering practices – they address different areas and complement each other.

Layout of the New Software Engineering Course

Table 6 shows the layout of the course. The course was divided into two parts of equal length and was designed with the insights from the previous chapters in mind. The two parts reflect the competence pyramid in Figure 1. Part one (weeks 1 to 7) lays the focus on building a strong foundation, i.e. the engineering practices. Part two (weeks 8 to 14) builds the second and third layer of the pyramid, i.e. the management practices and values. All practices were actively applied in a student project during part two.

For this course, the following learning target was defined using Bloom's taxonomy [19]: "After successfully attending this course, students have the necessary skills to develop software in an agile team. They can apply the most important agile engineering- and management practices and understand the importance of the agile values."

Week	Lecture	Workshop
1	eXtreme Programming Agile Manifesto	Installation IDE & Plug-Ins Coding Assessment 1
2	eXtreme Programming Version Control	Coding Assessment 2 Version Control Syst. (SVN)
3	eXtreme Programming Project Automation	Build Scripts (Ant)
4	Continuous Integration	CI (Jenkins Build Server)
5	Unit Testing	JUnit
6	Unit Testing / Mock Objects Clean Code/ Code Smells	JUnit EasyMock
7	Refactoring	Refactoring
8	Introduction to Test-Driven Design / Scrum	TDD, The Craftsman articles
9	Scrum	Agile Game Development (Sprint 1)
10	Scrum	Agile Game Development (Sprint 2)
11	Agile Estimating and Planning Planning Poker	Agile Game Development (Sprint 3) Agile
12	Metrics Agile Teams	Agile Game Development (Sprint 4) Metrics (EMMA)
13	User Stories Agile Principles	Agile Game Development (Sprint 5)
14	Demonstration of computer games	Agile Game Development (Sprint 6)
15/16	Preparation for examination: No lecture	Preparation for examination: No workshop

Table 6: Overview of semester plan

Part One: Applying Engineering Practices

- *eXtreme Programming (XP)*: In the first two lectures the students were given an introduction to XP. The XP practices and the Agile Manifesto were discussed. In the workshops, each student completed a coding assessment and was given feedback.
- *Version Control:* As a preparation for Continuous Integration, the concept of a version control system (VCS) was introduced. Subversion (SVN) was used as repository in the workshop. Some students suggested that GIT should rather be used than Subversion.
- *Project Automation:* Ant (Another neat tool) build scripts were introduced in the lecture and practiced in the workshops. Some students suggested using Maven instead of Ant build scripts.
- *Continuous Integration (CI):* With version control and project automation in place, the concept and benefits of CI were discussed. In the workshop, a CI-server Jenkins was configured.
- *Clean Code and Code Smells:* Clean code has had a marvelous effect on the quality and readability of student's code [9,10]. The students read most of the Clean Code book as part of the self-study.
- *Unit Testing and Mock Objects:* The concept of automatic unit testing was introduced. In the workshop, exercises with JUnit and EasyMock were carried out. These JUnit tests were added to the CI-server.
- *Refactoring:* Good understanding of automatic unit testing and refactoring are the basis of Test-Driven Design. A catalog of refactorings was discussed and practiced in the workshop.
- *Introduction to Test-Driven Design (TDD):* "TDD is hard. It takes a while for a programmer to get it." [17]. TDD is especially difficult to teach in the classroom. For that reason, the students were only given an introduction to TDD. In the workshop, the students worked through some of the craftsman articles [18]. One student gave

Items	excel-lent	good	bad	very bad
The content of this course is...	12	11	0	0
This course was divided into engineering- and management practices and agile values. How would you judge this concept?	12	11	0	0
How did the agile values come across in the lectures and workshops?	1	19	1	0
In the student project, you worked in a Scrum team of 6 to 8 fellow students. How would you judge this concept?	9	11	4	0
How would you judge the workshops in part one?	1	20	1	0
How would you judge the workshops in part two?	6	14	3	0

Table 7: Course evaluation

Items	Yes	No
Would you recommend this course to your fellow students?	23	1
Did you enjoy this course?	20	0

Table 8: How did you like the course?

"… the development of the computer game in a Scrum team."
"… that the material in the course was not only covered theoretically but I also had the opportunity to apply and deepen it in the workshops."
"… the practical relevance."
"… that the topics covered were interesting and important. I had the opportunity to practice the newly learned in the student project. That was great!"

Table 9: What did you like best about the course?

the following feedback: "Reading the craftsman articles really helped me to understand how TDD works."

Part Two: Applying Management Practices

- *Student project:* While the students were working individually or in small groups in part one, part two was different - the agile game was played in the classroom. In order to really understand how Scrum works, the students must be members of a "real" Scrum team. Since this is not possible in the classroom, the Scrum team was simulated in the student project. The goal of the student project was to develop a 2D computer game applying all needed engineering practices. The students worked in four Scrum teams of six to eight. Each team was free to decide what kind of computer game they wanted to develop. One student was voted ScrumMaster; the lecturer was the product owner. The teams completed six one-week sprints. Every week during the workshops, each team did the sprint planning, sprint review and retrospective coached by the lecturer. During self-study, the students developed the actual game. In the last week, all the teams could demonstrate a working game. In order to get a good start, the students were given an introduction to game development with Slick2D [20].
- *Scrum* was introduced in the lecture. Problems and questions, which had arisen in the Scrum teams where addressed in the next lecture and discussed in the plenum.
- *Pair Programming* was introduced ad hoc. The students were asked to pair with peers while developing the game.
- *Planning Poker:* Agile estimating and planning was introduced during the lecture and practiced in the Scrum teams [11]. User stories were estimated by playing planning poker [12].
- *Task board:* The functioning of the task board and burndown charts were discussed. For this course, an electronic task board was used.

Teaching Agile Values

Agile values are difficult to teach [13]. The approach in this course was to show the students, that these values are not just something the creators of the Agile Manifesto intended to give lip service to and then forget. They are working values. The concepts of agile values were introduced in the first part. Usage of the values was propagated in the second iteration through means like retrospectives, common code ownership or pair programming. Many discussions during the lectures and workshops tried to transport that message.

Student Feedback

In the last week of the semester, 24 students filled in an evaluation form (the items and answers are translated from German). An excerpt of the encouraging results is shown in Table 7.

In the planning phase there was some uncertainty as to whether the student project would falter due to group size and commitment of the individual member of the scrum teams. These fears were ungrounded. On the contrary, the students were exceptionally committed and delivered top quality computer games. The students were asked what they liked most about the course. In Table 9 are some statements, translated from German. The students were also asked what they disliked about the course. Nine students did not have any dislikes. Most of the students disliked the amount of work during the student project in the second

part. Many students suggested that the student project should be longer (see Table 10).

Evaluation and Suggestions

The quality of the students' work was measured twofold. On the one hand, the student project presentations, which included a demonstration of their computer games, were evaluated. On the other hand, the students had to pass a formal oral exam. The average grade was a 5.1 on a scale from 1 (very poor) to 6 (excellent). This was higher than expected. A systematic classification of the outcome quality remains to be done.

The experience from this course and input from students lead to the following suggestions:

Group dynamics are very important and therefore special attention should be paid to the way the Scrum teams are put together. The students should have access to a room, where they can meet for standups and have a wall for the task board. For this course, an electronic task board was used. Unfortunately, because of poor performance it did not meet our expectations.

Working only a couple of hours every week on the student project is not ideal. Many students suggested an intensive week instead. During this week, the students would only work on the project in the Scrum team. One semester is rather short for the material covered in this course. If the students had been familiar with engineering practices like unit testing, refactoring, build automation or clean code prior to the course, this time could have been used for test-driven development or additional iterations.

Further Work

Advanced practices like Behavior Driven Development (BDD) or Acceptance Test Driven Development (ATDD) were not covered in this course. Because of limited time, only an introduction to Test-Driven Development could be taught. Testing is a very important topic and should therefore be deepened in future courses. The same is true for requirements engineering, which was only partly covered in this course.

It is the authors' opinion that agile software development cannot be taught in isolated Software Engineering courses. A challenge will be the integration of agile development in other courses like programming, object-oriented analysis and design, algorithms and data structures, etc. Special attention needs to be paid to the fact, that agile software development does not work well together with big-design up front (BDUF) approaches. This could mean a shift from BDUF to emergent design as advocates of Scrum propose it. That said, further work is necessary on how agile development can successfully be integrated into the computer science curriculum.

"… too much work during the second part".
"… too little time for developing the computer game".
"… agile was praised too much. Negative aspects of agile were not or too little mentioned".
"… the electronic task board".
"… too little time for the student project, because of simultaneous projects in other courses".

Table 10: What did you dislike about the course?

References

[1] Version One. State of Agile Development Survey results. http://www.versionone.com/state_of_agile_development_survey/11/, 20.10.2012

[2] Martin Kropp, Andreas Meier, Swiss Agile Study - Einsatz und Nutzen von Agilen Methoden in der Schweiz. www.swissagilestudy.ch, 20.1.2013.

[3] Orit Hazzan, Yael Dubinsky: Why Software Engineering Programs Should Teach Agile Software Development. ACM SIGSOFT Software Engineering Notes 2007, Vol. 32/2.

[4] Bernd Bruegge et al: Agile Principles in Academic Education: A Case Study. 6th International Conference on Information Technology: New Generations 2009. ITNG '09.

[5] Vladan Devedzic and Sasa R. Milenkovic. Teaching Agile Software Development: A Case Study, IEEE transactions on Education Vol. 24. No 2. 2011.

[6] Andreas Schroeder et al. Teaching Agile Software Development through Lab Courses. IEEE Global Engineering Education Conference 2012. EDUCON '12.

[7] Viljan Mahnic. A Capstone Course on Agile Software Development Using Scrum. IEEE TRANSACTIONS ON EDUCATION, VOL. 55, NO. 1, 2012

[8] Rico, D.F., Sayani, H.H. Use of Agile Methods in Software Engineering Education. Agile Conference, 2009. AGILE '09.

[9] Robert C. Martin, Clean Code: A Handbook of Agile Software Craftsmanship, 2009, ISBN 0-13-235088-2

[10] Robert C. Martin, The Clean Coder: A Code of Conduct for Professional Programmers, Prentice Hall, 2011, ISBN 0-13-708107-3

[11] Mike Cohn, Agile Estimating and Planning, 2006, ISBN 0-13-147941-5

[12] Mike Cohn, User Stories Applied, For Agile Software Development, 2004, ISBN 0-321-20568-5

[13] Agile Manifesto. http://agilemanifesto.org/, 20.1.2013.

[14] Ken Schwaber, Mike Beedle. Agile Software Development with Scrum, 2001, ISBN 0-13-207489-3

[15] Kent Beck, Extreme Programming Explained: Embrace Change. Addison-Wesley, 2004 ISBN 0-321-27865-8

[16] Kent Beck, Test-Driven Development: By Example. Addison-Wesley, 2003, ISBN 0-321-14653-0

[17] Henrik Kniberg, Scrum and XP from the Trenches. How we do Scrum. An agile war story, 2007, ISBN: 978-1-4303-2264-1

[18] Robert C. Martin, The Craftsman, http://www.objectmentor.com/resources/publishedArticles.html, 20.1.2013.

[19] Benjamin S. Bloom, David R. Krathwohl (1956). Taxonomy of Educational Objectives: The Classification of Educational Goals, by a committee of college and university examiners. Handbook I: Cognitive Domain, New York, Longmans, Green.

[20] Slick2D - Open source 2D java game library: http://www.slick2d.org, 22.01.2013