Agile Project-Based Teaching and Learning

Dagmar Monett
Computer Science Dept.
Faculty of Cooperative Studies
Berlin School of Economics and Law, Germany
Dagmar.Monett-Diaz@hwr-berlin.de

***Abstract*—Agile courses in university settings aim to prepare students to face the ever increasing demands from the software industry, where Agile has become mainstream. This proves the teaching and understanding of Agile in such settings is of the utmost importance. This is why Agile is no longer just a part of the software engineering curriculum in Computer Science but a standalone course in most cases, though with increasing challenges for both faculty and students. This article presents yet another example case of the design, planning, development and evaluation of an agile project-based course. The reason for addressing the Agile teaching is twofold: not only are the Agile theory and practice taught and experienced in class, but also the teaching itself, and consequently the learning, has been adapted to changing requirements and priorities in each edition of the course. Making it project-based allows students to work with realistic projects through which they learn Agile more effectively, in collaborative and self-organizing teams. These insights, as well as settings and experiences over a total of 4 years, are addressed in this article.**

***Keywords*—Agile, eXtreme Programming, teaching, project-based learning.**

I. Introduction

There are lots of strong reasons for including Agile principles in CS education [1]. Positive experiences that go from *project-based* Computer Science (CS) courses using Agile [2, 3] over *Agile teaching* [4] to *Agile instructional design* [5] have had a common denominator: the practices, the values and the methods of the agile software development are essential; Agile is a current mainstream in the software industry [6] and educational environments are profiting from this, too. Meanwhile, project-based learning has proven to be very attractive in tertiary teaching: students learn the discipline via a realistic project, they pursue questions and connect them to activities that are part of the project, they construct knowledge and autonomously work towards a final product, as well as they master the curriculum standards with academic rigor [7].

The module *Project Management* is part of the CS education during the third semester at the Berlin School of Economics and Law (BSEL). By successfully passing this module, dual studies CS students can obtain 14 ECTS-credits,[1] which are assigned by considering the following proportion: a 20% of them goes to the sub-module *Project and Quality Management*, a 30% goes to the sub-module *Multidisciplinary Lab using Agile techniques*, and a 50% goes to the sub-module *Practice Transfer*, where students are at their enterprises and where they should apply gained knowledge in software engineering in general and in Agile and project management in particular. Credit hours, however, were never intended to be a measure of student learning, as Laitinen argues in [8]. She brings forward the argument that there should be found "what students are expected to –and actually do– learn", as well as the measurements to meaningfully assess what they have learned, not only concerning time-based units. By introducing Agile project-based techniques in CS assignments and by accurately defining both the learning goals and their evaluation forms, as it is further presented in this article, a positive step in this direction is achieved.

Much of the Agile courses in university settings have a common goal: to prepare students to face the ever increasing challenges in the software industry. Jaccheri and Morasca define in [9] five main roles that industry can play in software engineering education from the point of view of the university teacher: industry as students, as teachers, as researchers, as customers, and as former students. Three of these roles are well-identified in the mentioned module *Project Management*:

- Industry as teachers: the sub-module *Project and Quality Management* runs parallel to the sub-module *Multidisciplinary Lab using Agile techniques*. The first sub-module is taught by an industry specialist in close collaboration with the latter's teacher.
- Industry as customers: a real customer, who presents a problem to the students and who is available for consulting, is simulated in the Lab, if it is not possible to invite "a real" one. The concrete problem that is selected and the algorithm for solving it are also present in many industrial applications.
- Industry as former students: there are a Faculty Technical Commission and a Faculty Commission for Cooperative Studies at the BSEL both integrated by several industry partners, former dual studies students some of them, that discuss and approve the curriculum and other teaching and learning issues. Part of the faculty is composed of former BSEL students as well.

Two of the most important advantages of the program that prepare CS students for their further professional life are: firstly, students from the Faculty of Cooperative Studies are dual studies students and work in German companies from their first career's semester on. This means, they gain practical experience in real industry scenarios from the beginning of their studies on. Second, the sub-module *Multidisciplinary Lab using Agile techniques* (Lab using Agile, for short) provides them with several hard skills like specifying, designing, implementing and testing software, as well as communicating,

[1] European Credit Transfer and Accumulation System. One credit point is equivalent to 30 hours of study.

presenting, and working in a team, to name a few soft skills. Furthermore, both advantages successfully minimize new hires' common frustrations, as addressed in [10].

The Lab using Agile uses an interdisciplinary approach from the viewpoint of different cross-disciplinary topics addressed there. Perhaps these are reasons why the course has been favorably received by both faculty and students. Its careful design and planning, as well as its constant adaption to changing teaching and learning requirements has proven extremely effective in project-based courses. The remainder of this paper describes aspects for the Lab using Agile in detail.

II. Agile and XP techniques

One of Agile's most used methodologies is eXtreme Programming (XP), which has also been very popular in CS teaching [11–15]. For example, Stapel and colleagues propose in [15] a XP lab design property system for teaching a project-based XP course to CS master students, emphasizing in XP practices as part of a closed block course. Their work inspired the study summarized in this paper, which recommends a change from a weekly course to a blocked one. However, not only the course design, its type and the students' level, but also the blocks' duration, the XP iteration lengths, the team sizes, and the project content, among others indicators, differentiate their research from the one presented in this paper. Valuable insights from other works evaluating Agile in education environments also influence the findings presented here.

Pair programming is no longer extrinsic to CS education. In [16], for example, a case study concludes that pair programming is an effective approach for mastering computer programming together with cooperative learning principles. The authors extensively review the literature about the advantages and disadvantages of pair programming as a teaching-learning strategy, too. In [17], the authors additionally comment about the benefits of pair programming when practicing it in graduate software engineering class projects. Furthermore, several works have been published concerning both the strengths and weaknesses of pair programming but from the perspective of the Agile community.

The rest of the XP techniques are also introduced to the students in the Lab using Agile, both theoretical and practically. The students are, however, undergraduate students with little programming experience. In fact, they have only attended a few semesters at the university. Nevertheless, they learn quickly how to develop software with the aid of Agile, they solve a concrete real problem working in teams and they gain experiences by simulating a working day at an enterprise as part of the course project.

Differentiated supervision and guidance allow for better reactions to problems that might arise when introducing Agile or simply when working with others. In the Lab using Agile, individual and general coaching is offered as well. The faculty coaches individuals and teams in the course and is able to monitor progress and development anytime. Thus, continuous feedback can be provided to the students, to the teams and to the entire group. In reciprocation, students should be capable of presenting different stages of working software, and they should discuss with faculty in the role of (simulated) customers, acceptance criteria for their software products. In the coaching sections, it is expected that students come with concrete questions they have prepared in advance about any topic they need advice on.

TABLE I. Course schedule: teaching blocks and semester credit hours.

Block 1	**Block 2**		**Block 3**		**Block 4**
Day 1	*Day 2*	*Day 3*	*Day 4*	*Day 5*	*Day 6*
6 SCH	8 SCH (PC Lab)	8 SCH (PC Lab)	8 SCH (PC Lab)	8 SCH (PC Lab)	6 SCH
	16 SCH		16 SCH		
44 SCH					

TABLE II. Course schedule: teaching blocks and Agile cycles.

Block 1	**Block 2**		**Block 3**		**Block 4**
Day 1	*Day 2*	*Day 3*	*Day 4*	*Day 5*	*Day 6*
Syllabus Intro I	Intro II Planing game 1		Release 1 Planing game 2		Release 2 Conclusion
	Iteration 1 (Incremental teamwork)		Iteration 2 (Incremental teamwork)		

III. Course Schedule

Table I shows the course schedule for the Lab using Agile in teaching blocks and semester credit hours (SCH, 1 SCH meaning what follows 45 minutes of teaching time). The course is divided into four teaching blocks for a total of 44 SCH. Blocks 2 and 3 take place in a PC Lab. They are mainly intended for teamwork. In the Fall 2009 and 2010 editions of the course, three XP iterations were programmed for respective three product releases. However, in the Fall 2011 and 2012, only two XP iterations and their respective releases were planned, in response to the course appraisals administered at the end of the previous terms. More on this respect can be found in Section VI-B.

Table II shows the same course schedule but in teaching blocks and Agile cycles. Both *Syllabus* and *Intro I* at Day 1 conform Block 1 and refer to an introductory section, which states the purpose and goals of the course, as well as the theory about the algorithms selected to solve the customer problem. *Intro II* at Day 2 refers to an introduction to Agile and to XP. Days 2 and 3 are two continuous calendar days from Block 2, as well as days 4 and 5 are from Block 3. *Iteration 1* starts with *Planing game 1* and takes between three and four weeks until *Release 1* is accomplished, with only the first two days at the university. This similarly occurs for *Iteration 2*, whose *Release 2* takes place at the end of the course, at Day 6. The *Conclusions* are mainly based on the presentations of the final product releases and on the teacher's feedback concerning the projects as a whole. In [15], to name one crucial difference to this work, the block course has no interruption at all: the (very short) iterations are continuously located in the course time frame.

Incremental teamwork in blocks 2 and 3 means students become more independent while working in a team. Students not only do work incrementally on different tasks without interruption while planning and developing software: they also apply Agile techniques that make them more independent.

They progressively need lesser coaching from faculty for mastering activities that are more complex with time. In order to cope with these challenges, the course schedule includes more time for programming and less for other didactic exercises, also in a progressive way.

IV. Learning and Teaching Goals

Faculty should be aware of both the coarse and the fine-grained learning goals for a course, in order to break down those goals and to focus on the content to be taught. The former, the coarse-grained learning goals, are often defined in the curriculum in a general way. The latter ones help faculty to plan and to draw up in detail what students need to master and the ways of achieving and evaluating that. By defining thoroughly the fine-grained learning goals of the Lab using Agile, faculty creates the course syllabus without difficulty, and individual blocks and days are planed easier. This does not require a straightforward, additional effort for the conception of all these teaching materials, but the time saved later pays dearly the invested one.

The second block of the Lab using Agile is dedicated to the first experiences with the XP practice, especially at Day 2. The fine-grained learning goals of the second block (B2) for the firsts double credit hours (2 SCH each, i.e., $1\frac{1}{2}$ hours) are:

After completion of the second block, the students will be able. . .

B2.1 (2 SCH): *. . . to identify and to describe software requirements using story cards; to assess their priorities; to coordinate and to discuss their inclusion in the current iteration; and to plan and to schedule related activities for the first XP iteration.*

B2.2 (2 SCH): *. . . to meet and to participate in "stand-ups" or daily meetings; to develop software programming in pairs.*

B2.3 (2 SCH): *. . . to discuss and to formulate rules for working in a team; to discuss and to formulate rules for the work of several teams in a room.*

B2.4 (2 SCH): *. . . to develop software working in teams.*

Didactic exercises worked out in this block include organizational aspects that allow for better collaborative work when applying XP, since this is essential to Agile [18]. Rules for working in a team are then to be discussed by the students, for example, and each project group could present its set of rules using a flip chart in one of the sessions.

Teaching screenplays were used to better schedule the sequence of concrete teaching and learning activities to be included into a class, as well as the time required to complete them. They were planned using a sandwich structure, i.e., by combining passive and active learning units, and are like lesson plans or teaching worksheets that describe the teaching roadmap for a class or for part of a class in detail. For example, the teaching screenplay for the first double credit hour from block B2 is shown in Table III. It corresponds to the fine-grained learning goals defined above for the first double credit hour of that block, i.e., for B2.1.

TABLE III. Example teaching screenplay for the double credit hour B2.1.

90 min.	**Entry**	5 min.	**Start** – *passive unit* Welcoming (oral) Contents and time schedule (flip chart)
	Working phase	20 min.	**Content 1** – *passive unit* Motivation (oral) Learning goals (flip chart) Planning game (flip chart, blackboard) Story cards (blackboard)
		3 min.	**Brainstorming** – *active unit* Collect examples (plenum)
		20 min.	**Content 2** – *passive unit* Project description (hand outs) Project goals (blackboard) Project requirements (hand outs) Requirements for 1st release (blackboard)
		2 min.	**Introduce exercise** – *passive unit* Planning game: method, time management (oral)
		35 min.	**Knowledge transfer** – *active unit* Planning game 1st iteration (teamwork, coaching) Define story cards Set priorities Discuss realization
	Exit	5 min.	**End** – *active and passive unit* Questions, feedback (oral) Conclusions (oral) Short about the next double SCH, i.e., B2.2 (oral)

TABLE IV. Example teaching screenplay for a double credit hour with teamwork.

90 min.	**Entry**	2 min.	**Start** – *passive unit* Welcoming (oral) Goals and time schedule (flip chart)
	Working phase	83 min.	**Teamwork and coaching** – *active unit* Incremental software development (by students) Individual team coaching (by faculty) Questions, feedback (team-oriented)
	Exit	5 min.	**End** – *passive unit* Conclusions (oral) Short about the next double SCH (oral)

19 such teaching screenplays are needed for blocks 1 to 3, i.e., one screenplay as in Table III for each double SCH. However, much of them are only an outline like the one presented in Table IV. All teaching screenplays can be adjusted and adapted depending on the concrete class' rhythm when developing the course projects, which is just an expression of the Agile project-based teaching. An extra column could be added to the screenplays, too, for comments on self reflection and on self assessment after completing the scheduled exercises and activities.

V. Project Requirements

The general project description was formulated as follows: *Solve the traveling salesman problem (TSP) using a metaheuristic algorithm in the context of an XP project. Wanted is a software product with a graphical user interface (GUI) that includes menus and controls to define settings and that visualize results, as well as with a graphical window to show both the cities and the optimization process in real time.*

Students should use metaheuristics algorithms, like genetic

algorithms (GA) and ant colony systems (ACO), to solve instances of the TSP. They should test their programs using 2-dimensional, symmetric TSP instances of geographical problems from TSPLIB [19], as well as they should report both their findings and the software development using Agile in a research paper of at least five pages, following the guidelines for two-column conference proceeding in IEEE style.

Software requirements are defined by the customer (real or simulated) at the beginning of each XP iteration, depending on the focus the software development in that phase is centered around. Only those requirements related to the GUI development, for instance, are defined, specified, planned, and prioritized in the same planning game. Those requirements concerning the data and the algorithms to process them are defined in another planning game. Whether to start with the GUI or with the logic was discussed with the students. For many of them it was more important and attractive to have a working product with options and other components to present to the customer in the different releases, into which other functionalities could be added onto.

In Fall 2012, the first release, at the beginning of the third block (see Table II), was an "individual" meeting of each team with faculty playing the roles of *customer* and *coach*. The second release, in the last course's block, was a "public" meeting (all teams, in plenum), where faculty played both the *customer* and the *evaluator* roles. Each team presented a software prototype in the former, as well as it addressed the main aspects related to other Agile methods and techniques. In the latter, the final release, a formal oral presentation of about 35 minutes gave insights about the final product, about the project development, and about the experiences and lessons learned during the project completion.

Emphasis was also put on project management tools for collaborative work. The students had the opportunity, at least in the last two editions of the Lab using Agile, to test and to use several new tools (for them), like Redmine[2] and Trello[3], for instance.

VI. Course Evaluation

The composition and the size of the class, together with other information related to the last four editions of the course, are presented in Table V. The number of students answering a customized, anonymous questionnaire at the end of the semester is given in parenthesis for each course edition.

In the Falls 2009 and 2010, the course was offered weekly and there were a total of three XP iterations (and therefore, a total of three releases). No special didactic methods were applied at that time. In each of both editions, a different algorithm was considered to solve the TSP problems, i.e., ACO in Fall 2009 and GA in 2010. Students had difficulties especially when programming in the class, since the time available each week was minimal. They also had problems that prevented them completing their projects on time.

In the Falls 2011 and 2012, however, the course was divided in four presence blocks, as it is presented in Table I. Both editions of the course scheduled only two XP iterations, as derived from students' feedback in the former courses. The algorithm used for solving TSP was the same in both cases (i.e., ACO). Both editions included several special didactic methods not applied before, as well as a close team coaching by the professor, more intensive in Fall 2012. Additionally, the faculty was coached in Fall 2012 by an external training coach, expert in didactic in higher education.

[2]Redmine (at http://redmine.org) is a project management web application.
[3]Trello is a board-based collaboration tool. See more at http://trello.com/.

TABLE V. Characteristics of the last four course editions.

Fall	Group size	Female prop.	Weekly/ Blocks	Agile iter.	Algorithm	Special didactic	Special coaching
2009	30(30)	1	w	3	ACO	–	–
2010	30(30)	2	w	3	GA	–	–
2011	24(24)	–	b	2	ACO	++	+
2012	28(19)	2	b	2	ACO	++	++

A. Evaluating Learning

Each student can earn at most 100 points, which are then converted to a grade-point system in the German grading scale, as usual. A final student's grade is the team grade to which they belong. It is determined using a percentage system with 20% for each of the following areas: first release, second release, research paper, software program, and project management.

For assessing the releases and the team presentations, an evaluation form was designed by the faculty. It considers key components like presentation skills, content, timing, confidence, quality, and so on. The research paper was evaluated according to guidelines for scientific events. What to consider for both its content and structure was previously discussed with the students. Last but not least, the software program should satisfy all requirements, the teams should submit an executable version out of bugs, and the main software features and their functioning should be shown in the final presentation, without forgetting the project management aspects related to the project as a whole.

B. Evaluating (not only) Teaching

By the term's end, a questionnaire independent of formal faculty evaluations was administered to students. The questions catalogue with their descriptive scale values is shown in Table VI. The questions are grouped in four major topics, these corresponding to the course requirements in particular, to teaching in general, to how students learned, and to Agile.

Students could also provide an overall evaluation of the course, including what they liked the most, what they did not like at all, as well as further suggestions and comments.

VII. Results and discussion

Figure 1 shows a polar line chart with an area layer divided in four sectors that depend on the four general questionnaire topics mentioned so far. The question P is not included since it refers to different scenarios (two or three releases).

The plotted data are computed using the following formula,

TABLE VI. QUESTIONS CATALOGUE WITH DESCRIPTIVE SCALE VALUES.

Id.	Question	Descriptive scale values and index			
		4	3	2	1[a]
A	What do you think about the required time for the course	too high	normal	too low	abstention
B	How were the requirements concerning the course assignments/tasks?	too high	realistic	too low	abstention
C	How did you find the problem that was selected to be solved (i.e. TSP)?	motivating	neutral	dissuasive	abstention
D	How did you find the algorithm that was selected to solve the user problem?	motivating	neutral	dissuasive	abstention
E	How was the introduction on the course goals and topics?	very good	normal	very bad	abstention
F	How did the teacher/on-site customer respond to the questions, how was her feedback?	very good	normal	very bad	abstention
G	Do you feel as if you would have learned something during the course?	very much	normal	very little	abstention
H	How did the course form your interest on the working field?	motivating	neutral	dissuasive	abstention
I	Did you enjoy Agile practices, especially XP?	very much	normal	very little	abstention
J	Do you think you have improved your programming skills when participating in the XP project?	very much	normal	very little	abstention
K	And how about your social skills? Did you improve them?	very much	normal	very little	abstention
L	Do you think that using XP improves the productivity of small teams?	very much	normal	very little	abstention
M	Do you think that using XP improves the quality of the code?	very much	normal	very little	abstention
N	Do you think that Pair Programming speeds up the developing process?	very much	normal	very little	abstention
O	How did you find the planning game at the beginning of each iteration?	very helpful	normal	irritating	abstention
P	How was the division in two (Fall 2011, Fall SS2012) / three (Fall 2009, Fall 2010) releases?	excessive	adequate	insufficient	abstention

[a] The scale index with value 1 is reserved for abstentions, for each question, so that students can leave questions unanswered.

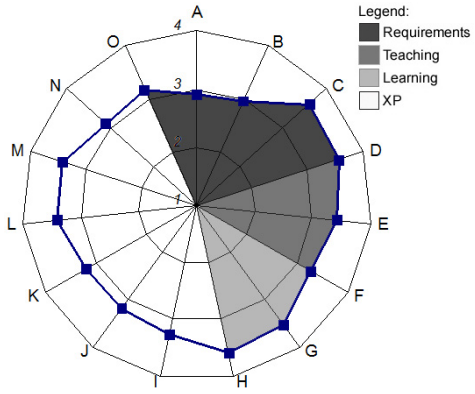

Fig. 1. Questionnaire results averaged for the four editions of the course.

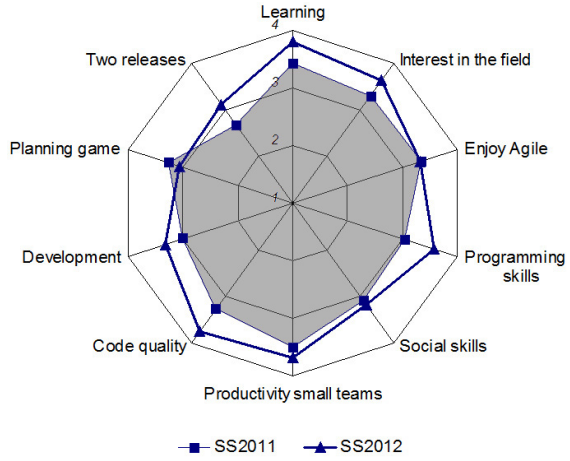

Fig. 2. Questionnaire results comparing Fall 2011 and Fall 2012 in detail.

which represents a weighted average for each question i:

$$y = \frac{\sum_{j=1}^{4}(5-j)\cdot v_{ij}}{N} = \frac{4\cdot v_{i1} + 3\cdot v_{i2} + 2\cdot v_{i3} + v_{i4}}{103}$$

N being the total number of students responses over the four years ($N = 103$) and v_{ij} being the sum of all responses multiplied by a scaling of the descriptive scale value j, for each question. For example, question A refers to the required time for the course and it has the descriptive scale values *too high*, *normal*, *too low*, and *abstention* (see Table VI). The number of total responses were 13, 70, 18, and 2 for each descriptive value, respectively. Thus, $y = 2.9126$ in the polar line chart for question A, which means that a substantial number of all students considered the required time as normal.

The rest of the plotted data can be read in a similar way: most students found the requirements concerning the course assignments (question B) to be realistic, the TSP and solving it with the selected metaheuristic (questions C and D) were motivating, and so on. All in all, the students' feedback was very positive in general, particularly regarding Agile.

Figure 2 shows a polar line chart with a polar area layer and a polar line layer comparing in detail some data for the Fall 2011 and for the Fall 2012, respectively. Only the questionnaire topics "how students learned" and "Agile techniques" are considered. In the figure, *Learning* refers to the question with identifier *G*, *Interest in the field* to *H*, *Enjoy Agile* to *I*, *Programming skills* to *J*, *Social skills* to *K*, *Productivity small teams* to *L*, *Code quality* to *M*, *Development* to *N*, *Planning game* to *O*, and *Two releases* to *P*, respectively, as specified in Table VI. The corresponding values are listed in Table VII, which includes the relative percentage of responses for each descriptive scale value, for each question, not including the abstentions for being irrelevant. Such details give more information than the weighted average when comparing both courses.

The main differences between the settings for Falls 2011 and 2012 concern the presence of female students (none in 2011) and the team coaching (more intensive in 2012), as it is presented in Table V. The questionnaire results, however, differ strongly in several aspects: almost all results for questions *G* to *P* show remarkable changes from Fall 2011 to Fall 2012. In the

TABLE VII. FALL 2011 AND 2012 COMPARED FOR GENERAL LEARNING AND AGILE DATA.

Question Id.	Fall 2011				Fall 2012			
	rel. %			glb. %	rel. %			glb. %
G	45,8	50	4,2	3,4	62,5	16,7	0	3,8
H	33,3	62,5	4,2	3,3	50	29,2	0	3,7
I	54,2	29,2	12,5	3,3	29,2	45,8	4,2	3,3
J	29,2	50	16,7	3,0	50	25	4,2	3,6
K	25	58,3	16,7	3,1	29,2	33,3	16,7	3,2
L	58,3	33,3	8,3	3,5	54,2	25	0	3,7
M	45,8	33,3	20,8	3,2	58,3	20,8	0	3,7
N	33,3	33,3	33,3	3,0	29,2	45,8	4,2	3,3
O	33,3	58,3	8,3	3,2	16,7	54,2	4,2	3,0
P	16,7	37,5	41,7	2,7	12,5	62,5	4,2	3,1

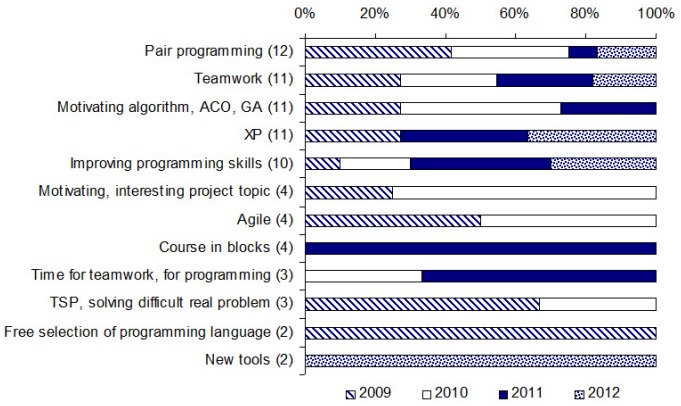

Fig. 3. Most mentioned positive comments.

latter, for example, most students feel they learned *very much* during the course (62.5%). One year before, more than half (54.2%) of the students considered learning between *normal* and *very little*. Similarly, for students in Fall 2012 the course is much more *motivating* than for their peers in 2011, they think their programming skills and the quality of the code are improved *very much* with XP, and two thirds find *adequate* the division in two releases (*insufficient* for 41.7% of the students in 2011). However, students from Fall 2011 enjoy Agile more (54.2%) despite more respondents selecting *very little* to describe the following Agile characteristics: speeding up the developing process with pair programming (33.3%), improvement of code's quality (20.8%), improvement of social skills (16.7%), as well as improvement of small teams' productivity (8.3%). These values were much more smaller or absent for responses from Fall 2012 and with descriptive scale *very little*.

Figure 3 shows the ten most positive impressions from the students, i.e., what they liked the most, from more to less frequent and after considering all four courses. Much of them refer to both Agile and XP. Pair programming was the most mentioned with a total of 12 occurrences. Both its benefits and practice were well accepted by the students. Working in a team and applying XP to implement a motivating algorithm was also very important for the students, as well as the chance to improve their programming skills in such a course project.

The students also had the possibility to mention what they did not like at all, as well as the opportunity to suggest changes to be considered in new editions of the course. Some typical responses were the following ones: it is too much work that has to be done for too few credits (there should be assigned more credits points for such a lab), the time pressure is too high (more time should be allocated for both programming and teamwork in the class), it is difficult to work in a room with too many teams at the same time (fewer teams should work in the same room).

The overall evaluation of the course in the four editions was as follows: About 80% of all students evaluated the course as *very positive* (18,45%) and *positive* (61,16%). A neutral evaluation was given by 18,45% of the students, mainly from the Fall 2011. Two students from the same year evaluated the course as *negative*, for a 1,94%. No student evaluated the course as *very negative*.

A subjective explanation of the negative results could be related to gender aspects, although no factual evidence is available. For years, usual comments between faculty staff, not only from Computer Science but also from the other three technical carriers at the BSEL, connect students' attention, participation and discipline in class to the presence or lack of female students. They argue that courses with female students have a better balanced classroom dynamic. The group attending Fall 2011 had no female students. A direct intervention was necessary several times to control both teamwork in the classroom and the discipline of few students. For that group, these aspects were the worst of all four editions of the course. It should be mentioned, in addition to this, that the teaching professor is female which is also infrequent in CS, at least in Germany. Furthermore, all female students from the other three years got the better grades, and this was also the case in other courses taught by the same female faculty. It is also worth pointing out that all females chose to do their two student research projects with this female teacher and their final grades were the highest possible scores. This supports Shaikh's conclusion in [20]: "the presence of female faculty in CS is also an important source of mentoring".

Another possible reason is the observed students' behavior during the course assignments and exercises. Most students were somehow resistant to participate in didactic exercises involving traditional methods other than the ones they use to work with while frontal teaching. Open feedback asked at the end of some blocks confirmed the argument that, when exercises were not directly related to programming activities for their projects, students were *wasting their time*. They could not see the potential advantages class games or student debates or think-pair-share might have on long-term learning. In Fall 2012, already knowing the difficult situations that arose in Fall 2011, students were instructed in advance about the goals and benefits of such kind of supporting exercises. Appropriate advise was also given by an expert coach. The working environment and the relations student–faculty were more relaxing and productive in 2012, in general.

The final grade in the module considers 30 points (from 100) for the Lab using Agile. The averaged final grades from all four editions of the course were:[4] Fall 2009, 27.82 (6); Fall 2010, 26.79 (7); Fall 2011, 28.92 (5); and Fall 2012,

[4]The number of teams is given in parenthesis. Each team is composed of 4 to 5 students, as a rule.

29.43 (7) points from 30. All in all, the grades were more than satisfactory: all students earned the required credits and the final grades were good despite the students' lack of participation and the difficult situations from the Fall of 2011. Most of the lost points were on scientific writing and not on the programs. The developed software programs were successful working products that satisfied the defined requirements and they were finished on time. Furthermore, the most XP values and practices were well understood by the students and were consequent applied during the project realization.

VIII. Conclusions

In this paper, the most significant differences between Agile weekly and block courses at the BSEL were presented. The combined use of all XP practices is very effective when developing Agile based-projects in these courses. Pair programming and whole team proved the most enjoyed by the students. However, students' engagement is higher in block courses because they have more time to concentrate and to participate in active learning tasks that need more time to complete. Students exploit the XP practices better when they work without interruption and when the teaching process is adapted accordingly. They are more able to improve their skills in planning and discussing, in analyzing and creating software, in evaluating and presenting results, as well as in working in teams in block courses than in weekly ones.

Since Agile's success in the software industry, it has been a constant in the CS curriculum at educational environments. Yet it is of utter importance not only how students learn Agile, but also how to teach it effectively. Teaching screenplays could help faculty in alleviating the conception and use of teaching materials. These roadmaps could describe the fine-grained learning goals of Agile teaching in detail. They proved to be very useful when used in Agile block courses.

Future work will be related to the introduction of other practices and techniques, for example from Scrum. The use of more tools to support the Agile development in the classroom is planned too. They should value individuals and interactions, working software, customer collaboration, and response to change, as Agile software development encourages.

References

[1] O. Hazzan and Y. Dubinsky, "Why software engineering programs should teach agile software development," *SIGSOFT Softw. Eng. Notes*, vol. 32, no. 2, pp. 1–3, March 2007.

[2] G. Perera, "Impact of using agile practice for student software projects in computer science education," *International Journal of Education and Development using ICT*, vol. 5, no. 3, pp. 85–100, 2009.

[3] A. Schroeder, A. Klarl, P. Mayer, and C. Kroiß, "Teaching Agile Software Development through Lab Courses," in *Proceedings of the IEEE Global Engineering Education Conference, EDUCON'2012*, Marrakesh, Morocco, April 2012, pp. 1–10.

[4] V. Razmov and R. J. Anderson, "Experiences with Agile Teaching in Project-Based Courses," in *Proceedings of the American Society for Engineering Education, ASEE Annual Conference & Exposition*, Chicago, Illinois, USA, 2006.

[5] D. Lembo and M. Vacca, "Project Based Learning + Agile Instructional Design = EXtreme Programming based Instructional Design Methodology for Collaborative Teaching," Dipartimento di Informatica e Sistemistica Antonio Ruberti, Sapienza Università di Roma, Italy, Tech. Rep. 8, 2012.

[6] , "7^{th} Annual State of Agile Development Survey," VersionOne, Inc., Atlanta, GA, USA, Tech. Rep., 2013.

[7] J. Thomas, "A Review of Project Based Learning," Prepared for The Autodesk Foundation, San Rafael, CA, USA, Tech. Rep., 2000.

[8] A. Laitinen, "The Curious Birth and Harmful Legacy of the Credit Hour," *The Chronicle of Higher Education*, January 21 2013, available online at http://www.scoop.it/t/higher-education-and-more/curate.

[9] L. Jaccheri and S. Morasca, "On the Importance of Dialogue with Industry about Software Engineering Education," in *Proceedings of the 3^{rd} Intl. Summit on Software Engineering Education, SSEE'2006*. New York, NY, USA: ACM, 2006, pp. 5–8.

[10] R. Conn, "Developing Software Engineers at the C-130J Software Factory," *IEEE Software*, vol. 19, no. 5, pp. 25–29, September/October 2002.

[11] A. Goldman *et al.*, "Being Extreme in the Classroom: Experiences Teaching XP," *Journal of the Brazilian Computer Society*, vol. 10, no. 2, pp. 4–20, 2004.

[12] K. Keefe and M. Dick, "Using Extreme Programming in a capstone project," in *Proceedings of the 6^{th} Conference on Australasian Computing Education, ACE'2004*. Australian Computer Society, Inc., 2004, pp. 151–160.

[13] M. Müller and W. Tichy, "Case Study: Extreme Programming in a University Environment," in *Proceedings of the 23^{rd} International Conference on Software Engineering, ICSE'2001*. IEEE Computer Society, 2001, pp. 537–544.

[14] A. Shukla and L. Williams, "Adapting extreme programming for a core software engineering course," in *Proceedings of the 15^{th} Conference on Software Engineering Education and Training, CSEE&T'2002*. Covington, Kentucky, USA: IEEE Computer Society, 2002, pp. 184–191.

[15] K. Stapel, D. Lübke, and E. Knauss, "Best practices in extreme programming course design," in *Proceedings of the 30^{th} International Conference on Software Engineering, ICSE'2008*. New York, NY, USA: ACM, 2008, pp. 769–776.

[16] E. Mentz, J. van der Walt, and L. Goosen, "The effect of incorporating cooperative learning principles in pair programming for student teachers," *Computer Science Education*, vol. 18, no. 4, pp. 247–260, December 2008.

[17] S. Xu and V. Rajlich, "Pair Programming in Graduate Software Engineering Course Projects," in *Proceedings of the 35^{th} ASEE/IEEE Frontiers in Education Conference, ICSE'2008*. IEEE, October 2005, pp. 7–12.

[18] K. Beck *et al.*, "The Agile Manifesto," The Agile Alliance, Tech. Rep., 2001.

[19] G. Reinelt, "TSPLIB - A Traveling Salesman Problem Library," *RSA Journal on Computing*, vol. 3, pp. 376–384, 1991.

[20] S. A. Shaikh, "Participation of Female Students in Computer Science Education," *Learning and Teaching in Higher Education (LATHE): Scholarship of Inclusive Curricula*, vol. 3, pp. 93–96, 2008.