Teaching Agile – Addressing the Conflict Between Project Delivery and Application of Agile Methods

Jan-Philipp Steghöfer, Eric Knauss, Emil Alégroth, Imed Hammouda
Chalmers | University of Gothenburg
firstname.lastname@cse.gu.se

Håkan Burden
Viktoria Swedish ICT
hakan.burden@viktoria.se

Morgan Ericsson
Linneus University
morgan.ericsson@lnu.se

ABSTRACT

This paper analyses the changes we have made in teaching agile methodologies, practices, and principles in four courses in order to address a specific dilemma: students need to apply agile methods in order to learn them, but when complementing our courses with applied content, we face the problem that students perceive the learning and application of agile methods as less important than delivering a finished product at the end of the course. This causes students to not apply theoretical process knowledge and therefore to not develop necessary skills associated with working with defined processes in the industry. Concretely, we report on our experience with teaching Scrum with Lego, removing formal grading requirements on the delivered product, emphasising process application in post-mortem reports, and organisational changes to support the process during supervision. These changes are analysed in the context of student satisfaction, teacher observations, and achievements of learning outcomes. We also provide an overview of the lessons learnt to help guide the design of courses on agile methodologies.

Categories and Subject Descriptors

K.3.2 [**COMPUTERS AND EDUCATION**]: Computer and Information Science Education; K.6.3 [**COMPUTERS AND EDUCATION**]: Software Management—*Software Process*; D.2.9 [**Software Engineering**]: Management—*Software Process Models*

Keywords

Teaching, Agile Methodogies, Project-based Learning, Scrum, Software Engineering Education

ICSE '16 Companion, May 14 - 22, 2016, Austin, TX, USA
© 2016 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ISBN 978-1-4503-4205-6/16/05...$15.00
DOI: http://dx.doi.org/10.1145/2889160.2889181

1. INTRODUCTION

Teaching agile methodologies, practices, and principles in software engineering education must allow the students to apply their knowledge and skills in the industry [2, 18]. However, the classroom setting often differs significantly from what the students later find at their employers. In particular, there is a fundamental difference between knowing theoretically about a process and being able to apply it in practice [4]. The Scrum community, e.g., maintains that "Scrum is simple, but not easy"[1] — while the different practices and aspects of Scrum are simple enough to be understandable intellectually, it is not easy to apply them correctly in practice and to leverage the advantages they afford. The same is true for other agile methods such as eXtreme Programming (XP) [25]. In comparison to "traditional" software processes, agile methods and principles are arguably much more hands on. A practice like planning poker or pair programming can not be understood on a purely intellectual level — these things need to be experienced by students to grasp them and to be able to apply them [16, 25].

However, we observe in a number of project courses, that theoretical communication of agile principles and isolated demonstration of practices and principles does not lead to their adoption in the projects. Instead, students tend to adopt a haphazard style of uncoordinated activity. We believe that the main reason for this is the fact that the *project content* overshadows the *agile methodology content*. Students focus on delivering a final hand-in (usually a software and/or an experience report) and on mastering the different programming languages, frameworks, and tools that are needed to deliver this product. This view is enforced by grading criteria that focus on the product, the application of the process being opaque to the teachers, and student support focused on the product. This makes it difficult to achieve learning objectives that focus on process knowledge. Similar problems do exist in the industrial adoption of agile methods [4]. The challenge is therefore to provide an environment in which the *process* by which the students arrive at the product is emphasised. At the same time, relevant technical knowledge and skills must still be imparted.

In this paper, we share our experiences with strategies to mitigate this dilemma that we applied over the course of one academic year. The research question guiding our

[1]See, e.g., https://www.scrumalliance.org/community/articles/2014/october/scrum-is-simple-but-not-easy

efforts was: **How can students apply agile practices in a realistic context without being distracted by technical difficulties of creating a product?** We found miniatures and simulated agile projects (e.g., agile/extreme hour [16, 17], Lego Scrum simulations [15, 18]) offering a solution that emphasizes agile practices over technical product and platform complexities on the expense of realism of project context. Complementing this approach, we found that emphasizing discussion of agile practices during sprint retrospectives over project progress helped students to focus on the desired learning goals. In addition, we changed grading criteria to move focus away from the delivered product.

The academic year we regard consists of two terms, starting in September 2014. We describe the status in four different courses within three different study programmes before the academic year started and the changes we have made during the academic year to achieve our goals. When appropriate, we accompany our observations and experiences with data collected from the students. This paper is the follow-up to previous work [1] in which we have described the fundamental problems and our first attempts to remedy the situation. We now report on the results of these attempts, using Schön's theory of the reflective practitioner [24] by reflecting-on-action. For each course, the individual challenges and how they have been addressed are stated while we retain a meta-perspective on our teaching and continuously discuss our overall strategy in what can be called a "guild of teachers". We also discuss the context of these changes, their effects, and the limitations of what we have done so far. We do not consider our work done — on the contrary, we plan to maintain (and with this paper: extend) our guild as a means of continuously improving our courses.

The paper is structured as follows: Section 2 highlights relevant findings from the literature. The context of this work and how we approached our joint effort is described in Section 3. The status of teaching agile in the courses we teach at the beginning of the academic year 2014/2015 as well as the changes we have made and the results we have observed during the academic year are analysed in Section 4. We discuss these changes, the effects we observed, and the lessons learnt in Section 5. We conclude with a summary and an outlook on our future work.

2. REFLECTIONS ON TEACHING AGILE IN THE LITERATURE

We distinguish two strands of related work: approaches to teaching agile in higher education and pedagogical frameworks that allow teachers to align course setup and the learning objectives. In the following, we describe the relevant approaches and how they relate to our setting.

2.1 Teaching Agile in Higher Education

Paasivaara et al. taught students global software engineering using distributed Scrum [19]. They emphasise the importance of frequent communication with the product owner to prioritise stories and demonstrate new features, that selecting an appropriate set of tools facilitating distributed development took time since the pure number of possibilities was over-whelming, and that the students became more confident in how to apply Scrum as the course progressed. Also from teaching distributed Scrum, Scharff et al. come up with a list of recommendations [22]. The list includes selecting a minimal set of tools to allow the students to quickly become effective, have a way to check the students' progress and make sure that the whole team is present during sprint planning. They also propose that the students prepare for the project by training Scrum using exercises such as Elephant Carpaccio [9] and the XP Game [20]. Scrum as a whole can also be taught as a game [18]. we have positive experiences with Agile Hours [16] and Extreme Hours [17] to help explain and train the planning game, and avoid wasting time on the basics during the first planning game.

Schneider and Johnston [23] state that it is not possible to get students to work co-located and continuously in most university settings because of disjoint timetables and lack of suitable laboratories. Many approaches instead concentrate on a limited set of practices that can be implemented in a typical course setting. For example, Astrachan et al. [3] attempted to create a lecture for eXtreme Programming (XP), which concentrated solely on, e.g., pair programming and refactoring, and Johnson and Caristi tried to integrate XP with software design [13] in their classes. While it is possible to cover for example all XP practices in a block course setup [25], the practices must still be adjusted to suite the limited time in course work.

Various suggestions for setting up a realistic environment for agile project work in class exist, such as Holcombe et al. [12] who choose to simulate a whole software company. Such realism is important, since agile courses need to convey the objectives practically. Many properties of agile projects must be experienced in order to appreciate their advantages and identify pitfalls that are not evident in theory [16].

Babb et al. [4] identify four barriers that impede learning during an agile project: having multiple projects or goals, the pressure of having to deliver each sprint, customer involvement limited to certain roles, and organisational culture. Each barrier has an effect on learning: developers did not share knowledge regarding the product, practitioners did not have enough time to reflect on their practices and how these could be improved, no direct access to the customer's expectations, and a lack of collective sharing of experiences and knowledge. These aspects play a role in our teaching challenges. For instance, we provide low customer involvement in the project courses, leaving the teams to define the details of the product themselves. At the same time, the organisational culture, i.e., the environment that is provided by the course setup, the grading criteria, and other factors favour the product over the process.

2.2 Pedagogical Framework

In their TPCK framework, Koehler and Mishra [14] emphasize that different knowledge areas must be covered in high-quality teaching: *pedagogical knowledge*, domain specific *content knowledge*, and *technological knowledge* about relevant technological aids for teaching and learning. The intersections of these knowledge areas are especially significant. In the context of this paper, *content knowledge* includes knowledge about principles and practices of agile software development as well as knowledge about agile development methods based on these principles and practices. *Technological knowledge* includes programming languages, source code management systems, and product related software development knowledge, e.g., about mobile phone platforms such as Android and iPhone. *Pedagogical knowledge* is the focus of this paper and is discussed below.

The other important theoretical concept is constructive alignment [6]. According to this principle, students construct meaning from what they learn, while teachers deliberately align teaching activities with intended learning outcomes. The goal is to optimally align learning and teaching activities with clearly specified learning goals, well designed assessment criteria and feedback to the learner. In the context of our courses, it is even more difficult to do so as in more engineering focused courses, because the applicability and success chances for practices and methods depend on context and the relationship is not completely understood in practice and research. To support beneficial learning outcomes, we need to emphasize group work as well as development towards a product. With respect to the constructive alignment, we thus need to address these aspects both in the setup of the courses and in the way we assess the students' performance. This can lead to an overemphasis of development work, if it is not accompanied with particular activities and artefacts that encourage reflection on agile methods, both during teaching and examination. Individual experience reports can encourage students to plan their project work in a way that allows for relevant experiences to happen and to reflect on them accordingly.

The case method [11] is one option to make real world agile problems more tangible during theoretical lectures. This would include discussing a realistic or real-world scenario in class. Providing a miniature or simulation can create such a case to which the theoretical knowledge can be applied.

3. TAKING ACTION: TOWARDS MORE FOCUSED AGILE TEACHING

This work was conducted at the joint Software Engineering Division of the Department of Computer Science and Engineering of the University of Gothenburg (GU) and Chalmers Technical University. Apart from courses in different programmes at both universities, it offers an undergraduate program in Software Engineering and Management and a Master in Software Engineering. Both programmes emphasise project-based learning and thus allow students to experience work in group settings with complex case studies and fixed deadlines. A number of these courses either include agile practices in their learning objectives or make use of them for the project work. In addition, the Software Engineering division provides courses for other programs.

As teachers at the Software Engineering division we are responsible for teaching agile in four courses in three contexts – *Software Processes* and *Software Architecture Project* at the Software Engineering and Management undergraduate program; *Software Engineering Project* for various engineering and IT programs; *Agile Development Processes* at the Software Engineering master program (cf. Table 1). Apart from Scrum, a number of agile practices are part of these courses and mentioned when appropriate.

The authors of this paper are the responsible teachers for the mentioned courses. At the beginning of the academic year 2014/2015, we started meeting regularly as a "guild of teachers" to discuss the challenges we face in our courses, possible mitigation strategies, and new ideas we received from colleagues and published papers. These discussions led to a number of changes that we introduced in our courses. By not regarding the courses in isolation but discussing the overarching issues within the programmes and aligning our understanding of our teaching, we have started to develop a common understanding of teaching agile and the possibilities and limitations of certain approaches within their context in the study programmes and in our common teaching agenda.

Since each course has a specific setup and specific learning objectives, there were no solutions that fit all the courses. However, they follow common threads as discussed in Section 5. Whenever we gained new experience, we came together and discussed our new knowledge. In this way, we performed a rather informal action research approach [8] in which the planning and reflection steps were done in a group whereas the act step was conducted by the responsible teachers in their respective courses. The changes that were made to the courses where few, but very concrete and targeted. Reflection was supported by our own observations as well as data from the students where appropriate.

4. ALIGNING PROJECT EXPERIENCE AND AGILE LEARNING OBJECTIVES

We report on the four courses to which we applied improvements. Each course is detailed with its specific setup, its challenges, the changes made, the perceived effect, and the plans for the next iteration of improvements. Depending on the available data, we use different approaches to describe the effects of the changes.

4.1 Software Processes

Course Setup. The Software Processes course is part of the curriculum of the first term of the Software Engineering and Management undergraduate program. The main objectives are: to enable the students to describe different processes and lifecycles; explain the differences between traditional and agile methodologies; describe the roles and activities in a software process; and explain why things do not always go according to plan. These objectives were addressed by theoretical lectures. Different lifecycles were introduced, their advantages and disadvantages were discussed, and agile principles were mentioned. The examination asked students to write an essay where they describe a software process that fits a certain software development scenario. The course did not have a strong practical element; in particular, the students had no frame of reference in which the theoretical content could be discussed. Since the students are in their first term, they have no experience of developing software in a team, let alone how software development in the industry is structured.

Challenges. Since the Software Processes course is the first time students are exposed to ideas of a structured and documented process, it is important to make sure that these ideas are clear and can be used in future projects. The missing frame of reference, however, prevents this since the students can not refer any of the theoretical knowledge to practical experiences. That causes issues later on, when they are asked to work according to a process in their project work, as witnessed in the first-term project and the Software Architecture Project, described below. Since knowledge from the Software Processes course was dismissed as largely irrelevant in practice and has not been solidified on the basis of actual experience, it is largely forgotten when it would be helpful. This is evident in the project course reported on in Section 4.2 where the process knowledge was not applied by the students correctly.

Table 1: Course Overview

Course	Product	Platform	ECTS	Duration	Goals
Software Processes	None	N/A	3.5	5 wks	Processes
Software Architecture Project	Apps	Social Media APIs	15	20 wks	Distributed Systems
Software Engineering Project	Apps	Android & Git	7.5	10 wks	Software Engineering
Agile Development Processes	Apps	Android, Git & Pivotal Tracker	7.5	10 wks	Agile Methods

Changes. Our attempt to remedy this situation was the introduction of a Lego Scrum simulation [15], held in the first weeks of the program. Using Scrum, the students were asked to develop a Lego city. Using Lego allowed students regardless of their technical skills (e.g., programming) to participate in the process productively as context knowledge is in focus. It also allowed to demo and review tangible results on an integration area. Tangible results facilitate understanding and make learning more playful [21]. Using Lego also meant that sprints could be confined to 18 minutes each which allowed each session of approximately 4 hours to include four sprints. The sprints were kept short to force the development teams to plan their sprints and to realistically estimate what can be done within a tight schedule.

We divided the students into four sessions, accommodating 14 to 21 students each, split into 3 or 4 development teams. The students had no previous exposure to Scrum. Each session started by outlining the fundamentals of Scrum in terms of backlog, roles and multiple iterations; we described the core activities of a sprint and introduced the role of the Scrum master. Planning poker [10] was also introduced to estimate the effort of a user story. Based on this, the teams had to define their velocity. The students were then handed a bag containing 40 litres of Lego. Each sprint was followed by a product review, during which we gave our feedback on the sprint, and a sprint retrospective that each team conducted on their own.

A total of four sprints were performed with a break between the second and the third sprint. We gradually introduced complications. In the second sprint, we disabled key contributors of each team by telling them they were "sick" for a couple of minutes and not allowed to build or talk to their team mates. In our capacity as product owners, we also tried to introduce new requirements during the sprint. In the third sprint, we added new user stories to the board and tried to disrupt progress of the team again. During the fourth sprint, the teams were left alone in order to allow them polish the product as a reward for their efforts. We changed the product backlog by incorporating bug fixes.

The general setup of the Lego Scrum simulations is similar to the one described in [18], but our simulations are less structured in order to allow the students to take responsibility for the sprints. In addition, the simulations are focused on creating learning opportunities for students that had no previous exposure to Scrum or other software development processes. We therefore emphasise inter-team collaboration, communication with all stakeholders, and let the students structure the sprints themselves.

The experiences from the simulation are referred to as a case [11] throughout the course whenever connections between the theoretical material and the practical experience are beneficial. The necessary frame of reference is thus established and the connections that are pointed out are anchored in the students' own experience.

Key Change
– Added Lego Scrum workshop as a way to provide practical experience to anchor theoretical knowledge

Effect. The *students spontaneous response* to this simulation format was enthusiastic. During a brief oral evaluation session at the end of the sessions, students were very pleased with the hands-on experience and working with Lego. On the other hand, they admitted to having felt stressed and feeling like the sprints were too short. This was, however, intended as described above.

The *course evaluation survey* showed that most students were very positive about the simulations. The students felt that they provided a hands-on way to engage with an agile methodology. As one student reports: "The SCRUM simulation exercise was great. It really helped me understand the idea behind agile processes as just reading about them isn't very effective." However, as noted in [18], enthusiasm alone does not mean that the students actually retain knowledge and are able to apply it in a project. A course were the Lego simulation was followed up by a project is reported on in Section 4.3.

Next Iteration. While the Lego Scrum simulation was successful in many ways, it did not cover some of the relevant Scrum practices. In particular, task breakdown and estimation was not covered well and it was obvious from observation that students struggled with this part. In other implementations of the simulation — such as [18] — a stricter structure for the sprints puts the focus on planning. However, these implementations require previous Scrum knowledge that the students of this course does not have. Instead, we intend to augment the Lego Scrum simulation with an *additional exercise* focused on task estimation, using not only planning poker but other agile estimation techniques, and task breakdown, using, e.g., the Elephant Carpaccio game [9], in which students need to create small "vertical slices" of user stories. In addition, there will be a *dedicated follow-up lecture* on the Lego Scrum simulation with hints on how the learned techniques can be applied in projects.

4.2 Software Architecture Project

Course Setup. In the Software Architecture Project the second year students of the undergraduate program practice their theoretical knowledge of distributed systems. The students work in teams of six to eight, expected to organize and conduct their teamwork by applying Scrum. In addition to the course teacher, student supervisors are employed to help students with the technical challenges of the project. Student groups are free to propose a concrete software application within a pre-set theme, where themes vary between course instances. Examples of prior projects include games, cyber-physical and business intelligence systems. Regular bi-weekly meetings are held to monitor the progress of the projects. An oral examination session is organized to assess the quality of the product developed and the extent to which students have followed Scrum as a development method. In addition to product delivery and process conformance, students are required to hand in a Software Architecture Document that provides a technical high level description of the system and a blueprint of the development activities.

Challenges. The Software Architecture Project is the first highly complex software development project course in terms of what the students are expected to deliver. This is due to the fact that the software developed needs to be a distributed multi-tiered system that is composed of a back-end component, a business layer and front end part. The developed product should also address strict fault-tolerance requirements. On the technology side, project teams are required to use Erlang as a programming language in implementing some of the system components. Furthermore, it is mandatory that teams hand in a well-written Software Architecture Document towards the end of the course.

We have observed a few problems in relation to the topic of this paper. First, students within a team tend to divide themselves into topic experts where each member is in charge of completing a specific part of the project. For example, some students focus solely on front end development while others devote their effort to developing the Software Architecture Document only. This leads to a situation where most students did not build a good understanding of what a distributed system is and did not learn about the technical challenges involved in developing a distributed system. The role-based development leads to poor interaction and communication between team members as each student focused on the problem at hand. In many cases, students found it very hard or even failed to integrate the different parts together, which heavily compromised the delivery of the product. Poor learnability, low levels of team interaction and delayed integration are good indicators of a dysfunctional team or anti-agile practices.

Changes. The organizational setting of the course has been changed so that each student team is assigned a student supervisor, who is typically someone who successfully completed the same course in previous years. The student supervisor acted as a product owner. In addition, each team was asked to select a Scrum master. No specific development roles were given to the team members. In contrast, they were required to work collaboratively on different parts of their projects. The course responsible acted as both Scrum coach and high-level project stakeholder. Once the project idea was set, students were asked to come up with a product backlog that is organized in terms of functional requirements. The product backlog as well as requirement priorities were then verified by the product owner.

The length of a sprint has been strictly fixed to two weeks. In order to organise their work during the sprints the students were free to choose their daily Scrum meeting strategy and a Scrum tool of choice. Sprint reviews, retrospectives, and planning meetings were held as two hour sessions every second week. Meeting place and time were also strictly fixed. The sprint reviews were attended by both the student supervisor (i.e., product owner) and course responsible (i.e., high level stakeholder). In addition, students were free to arrange meetings with the product owners in the middle of sprints, if needed. Students were required to hand in sprint retrospective reports after each retrospective meeting. The report needed to reflect on what the team has learned in the expired sprint, what has worked well during the sprint and what could be improved during the next sprint. The students were also asked to keep up-to-date project technical documentation (i.e., Software Architecture Document) instead of postponing it to the last sprint.

Key Changes
– Emphasized the role of product owner
– Organized all activities as backlog items
– Fixed sprint duration to two weeks
– Assigned students to different kinds of tasks across sprints

Effect. The course had 14 project groups that all managed to deliver an acceptable and in most cases an excellent software solution by the end of the course. The documentation parts which consisted of the retrospective reports and the software architecture documents were mostly satisfactory. To evaluate the process issues, we *collected observations* during the bi-weekly meetings. Our notes show that students have *a better understanding of Scrum* and implemented different *agile practices to a good degree of satisfaction.* We also noticed that most student groups started to hold *knowledge sharing sessions* where they taught each other specific topics they were in charge of. Furthermore, we conducted a *post-mortem examination* of how well students worked with agile/Scrum values, principles, and practices. A total of 40 students responded to the survey (about 50% response rate). 80% of the students experienced aspects of *respect, commitment, focus, courage, openness, self-organization*, key values of Scrum. To a lesser extent about 60% reported that empirical process control and timeboxing were highly relevant in their projects. Concepts and practices such as *project vision, product backlog, sprint backlog, user stories, sprint review, sprint retrospective, sprint planning, and the definition of done* scored very high (between 75% and 90%). Other practices such as acceptance cards, burndown charts and daily stand-up meetings were less relevant, scoring 40%, 45% and 50% respectively. A good number of students (63%) tried *pair programming* which shows increased levels of collaboration and interaction. When asked whether they would *consider using Scrum in another project course* more than 90% of the students agreed.

The effects of our changes show that we have achieved a better balance between product and process requirements compared to earlier editions of the course.

Next Iteration. The post-mortem survey shows that the course can still be improved to help students *experiment better with a number of agile practices* such as daily standup meetings, story estimation, and measuring team velocity. There is a problem with the spirit of *"inspect and adapt"*. In the next iteration we will ask students to collect and maintain *qualitative and quantitative data* to empirically assess their progress. We noticed that students do not systematically run retrospective meetings to reflect on how well expiring sprints went. In fact, they often report that everything is working well and there is nothing to improve. However, when forced to reflect, a lot of improvement were identified, to their great surprise. We plan to enforce *continuous retrospective* where students can give feedback on each other's work on an almost daily basis rather than waiting for the retrospective meeting held at the end of each sprint. We argue that those improvements would have a positive impact on product and documentation quality.

4.3 Software Engineering Project

Software Engineering Project is a second or third year course taken by students from the Computer Engineering, IT, and Industrial Economy (specialisation in IT) master programs. The course is given twice a year, during the spring and autumn terms.

The course is centred around a project where students develop an Android app in teams of five or six. The lectures serve three purposes: i) to introduce relevant theory regarding the specification, implementation and testing of software, ii) to support the project by explaining relevant technologies and Scrum, and iii) to illustrate industrial praxis by guest lectures. The theoretical topics are further explained by the course literature while the practical elements are supported by links to forums and tutorials.

The course is assessed by the user experience of the app (e.g., customer value, GUI, user stories and documentation) (30%), the implementation (i.e., code, system and unit tests, developer documentation, design artefacts) (40%), and their process as described in a post-mortem report (30%).

Challenges. The scope of the course syllabus is a challenge when teaching Scrum. Scrum is an important aspect to learn, but so are concepts such as testing and architecture. The project must be complex enough to require that students learn and apply software engineering concepts using the tools and techniques of the field. This complexity, especially on a technical level, becomes a challenge for the students since there are many new concepts to learn. Many students have no previous experience with version control systems or the Android SDK. These take time to learn and are often viewed as more important than Scrum since they are directly connected to what the students perceive as the goal of the course, i.e., to deliver a working app. The students also find it difficult to apply the theoretical Scrum lectures to their own project.

The size of the course is another challenge. Approximately 175 students attended (autumn 2014), so there were 29 teams to coach and assess. The large number of teams and one-week sprints made it difficult to observe how the teams work. We had to rely on student supervisors to be able to meet every team once per week. Based on our experience, student supervisors often focus on the technical challenges. So, even if we consider coaching an important way to work with Scrum, we had limited opportunities to do so during the course. The spring iteration of the course is taken by fewer students; 54 students (nine teams) attended the course in the spring of 2015.

Changes. Since there are fewer students in the spring iteration, we decided that this was the best time to implement three major changes to improve how we teach Scrum. The first change was to introduce a Lego Scrum simulation [15] to give students a hands-on experience with Scrum. The simulation was followed by a lecture where the experiences were given a theoretical framing. It was trivial to introduce the simulation, given the knowledge gained from the Software Processes course (cf. Section 4.1).

The second change originated from our newly gained pedagogical knowledge. In two different lectures we emphasised the process in the overall team assessment. We also added a lecture were we as teacher *reflected on and verbalised* [7] our processes for developing and teaching the course.

The third change was to turn the weekly supervision into a sprint meeting with acceptance testing and Scrum retrospective. All technical supervision was moved to an introductory workshop on getting started with Git and Android, and dedicated technical supervision sessions throughout the course. The teachers were responsible for the weekly sprint meeting where, e.g., the relationship between product and process was discussed in addition to feedback on the product increment and recommendations for how to carry out the next sprint. We relied on student supervisors for technical supervision to clearly seperate that knowledge, w.r.t. persons and occasions, from the sprint meetings.

Key Changes
- Added a Lego Scrum workshop
- Emphasised the importance of the process in the course assessment
- Separated technical supervision from the sprint reviews and retrospectives

Effects. We analysed the post-mortem reports, the responses from the course evaluation survey, and feedback from the course evaluation meeting to evaluate the changes. Table 2 provides a summary of the course evaluation surveys. The response rates are too low to determine whether the changes are statistically significant, but the numbers provide some indication. All answers are given on a five-graded Likert scale, ranging from *Poor* (1) to *Excellent* (5). The statement regarding workload is an exception, where 1 represents *Too low* and 5 *Too high.*

All teams experienced that adapting Scrum is a learning process. Four teams mentioned that the Lego simulation does not cover all aspects of Scrum, so it is not possible to directly map their experiences to the project: "It was also a learning process to begin working with the SCRUM approach, which the group took for granted that they would be able from the outset, but that turned out to have a certain learning curve". A fifth team reported that "We learned from the Lego exercise that we shouldn't take the more demanding features to start with since it is better to have something done completely rather than just nearly." Two teams explicitly stated that they found the sprint reviews an important opportunity to learn, "The continuous reflection on what has been done, what needed to be done and what worked or did not even gave rise to a major learning opportunity for all team members."

The students do not expect change. Since the students decide their own project they also own the product backlog. One team found that the lack of external influence allowed them to drop Scrum and only focus on agile practices such as pair programming and continuous integration. Four of the teams explicitly stated that there was no need for Scrum roles; "We opted out of these because we did not think it would add something when we do not have a proper customer of the app." Three teams even concluded that they owned the product and therefore also the process. As there were no external stakeholders to consider, "in the later sprints, we only used user and fuzzy stories". Since the need to reprioritise and refine epics was driven by the team, it was ultimately avoided due to time constraints.

More suitable way of teaching software engineering. The survey indicates that the students' overall impression of the course increased as well as their assessment of the teaching methods. The introduction of the Lego simulation had a positive impact. One student remarked that "The Lego exercise was good, some lectures less so." Finally, the course

Table 2: Student evaluation of the Software Engineering Project.

Statement	2014	2015
Overall impression of the course	3.3	3.92
Suitable course prerequisites	4.15	4.15
Appropriate teaching methods	3.4	3.77
Reasonable course workload	3.0	2.69
Response rates	12/34	13/54

workload was balanced in respect to the number of course credits and during the evaluation meeting the students remarked that the workload is not a problem when you own both the product and the process. The unchanged score for the suitability of the course prerequisites is in line with keeping the technological platform constant.

Next Iteration. *Each team will plan for implementing Scrum.* During the course evaluation meeting, a student asked for an exercise on how to adapt the experiences from the Lego simulation to the project setting. While the idea is sound it is difficult to find the time for another exercise or simulation. Instead, the teams will formulate a plan which is then a benchmark when evaluating their learning and process in the post-mortem report.

External stakeholders will make the process real. Since sprint meetings will be a challenge with 30 instead of nine teams, we cannot participate in all sprint meetings. One way to increase the number of teachers is to use external stakeholders that act as customers to the teams. External stakeholders can provide feedback on product increments and what do to next, but generally not act as Scrum coaches. So, in this aspect the teams will be worse off compared to the spring iteration. However, based on the post-mortem reports from the last iteration, it seems that the lack of external stakeholders lets the teams forsake a systematic process. The idea is that the lack of coaching might be mitigated by the demand for a more systematic process when external stakeholders are introduced.

4.4 Agile Development Processes

Course Setup. The Agile Development Processes course is a mandatory course for graduate students at the software engineering programme but is also attended by students from other programmes, e.g., industrial economy, interaction design, and phd programs. Intended learning outcomes include the Scrum process, but also a broader perspective of agile software development practices and principles.

To align theoretical content delivered in lectures and practical content explored in projects, the 8 week course is organized in three sprints. In the first sprint, up to 2 lectures per week provide the necessary overview and initial training in agile practices. Two (optional) tutorials on the *platform* of the course help to mitigate differences in knowledge of students with respect to github, android, junit, and software testing. In the second sprint, significantly less lectures (usually: two industrial guest lectures) allow students to focus on (partially supervised) project work. The easter break falls into the end of this sprint, allowing for additional project time. In the third sprint, there is again an increase of lectures with a wider range of advanced topics (large-scale agile, agile vs. plan-driven, and current research topics).

Five to six students work in each project with the objective to develop a mobile app. The teaching team acts as customers and coaches during the sprints (initially one week), with scheduled, partly supervised, working hours. Each sprint ends with a customer acceptance test performed by a member of the teaching team, checking what functions and features the group had implemented in the last sprint. The time spent on the acceptance test is constraint by resources such as availability of rooms and scheduled times (centrally by University), allowing only for ten minutes per group and acceptance test and leaving little or no time for feedback on the applied process.

Challenges. Course evaluations from the years 2011 to 2014 indicate that students enjoy the technical orientation of the project. While the students have rated the course quality as high in course evaluations, there has been a negative trend in the ratings, from above 4.0 to below 4.0 on a scale from 1.0 to 5.0. The trend can not be explained by any one factor, but we perceive that the technically focused project may have contributed since it splits the students' focus between the need to develop and deliver a functional mobile application and learning of agile methods, principles, and practices. By overemphasising product deliver over applying the agile principles and practices, students might not get the deeper understanding of how the agile practices are linked and how they connect to the agile principles.

Changes. To mitigate the perceived split of attention, several changes were made for the Agile software development course in 2015. Firstly, inspired by the Lego tutorial, one lecture in the beginning was replaced by a new miniature project to give the students a holistic trial run of agile methods and practices. Within a 90 minute slot the 70 students were split into random groups. Each group had customers and developers. In three iterations, customers would see a sketch and then in increasingly agile manner, make the developers reproduce the sketch with pen and paper.

Secondly, all formal requirements towards the developed app's functionality was removed from the assessment criteria. Instead, emphasis was put entirely on the students' use of predifined agile practices. In this case, we chose the twelve classic practices of XP [5, 25], since the main Scrum practices were already enforced by our organization of the project work.

In addition, we introduced a new sprint retrospective in addition to the (functional) acceptance test, where we checked and discussed the use of agile practices. To allow this, we increased sprint length from one to two weeks and acceptance test length from 10 min to 20 min. The discussion of agile practices allowed the teachers to see what practices were being used, how they were applied, and which challenges the students encountered with each practice.

Key Changes
– New agile mini tutorial.
– Removed learning objectives with respect to achieved functionality in project.
– Added retrospective to acceptance test to check for application of agile practices.
– Extended sprint length from 1 to 2 weeks.

Effect. The results of these changes are non-trivial to quantify, but based on collected information from the course and the course deliverables some observations can be made. *Students' perception about the course* was improved compared to the previous year, with an increase in 0.3 points

Table 3: Adoption of agile practices among the 11 groups in the two relevant retrospectives.

	R1	R2	Trend
TDD	0	7	↗
Planning game	8	8	
Customer on site	9	8	↘
Pair programming	11	11	
Continuous integration	8	11	↗
Refactoring	8	9	↗
Small releases	11	11	
Simple design	11	11	
Metaphor	5	7	↗
Collective code ownership	9	11	↗
Coding standard	9	10	↗
Sustainable pace	10	10	

from 3.7-4.0 on a 5.0 scale. However, whilst the overall perception improved, individual students still mentioned the course as being too technology focused. Hence, some students experienced that their group members, despite the focus on the process, spent more time programming than committing to using/learning the agile development process as expressed by the following quotes from the course evaluation: "…Despite telling us that the application was not important people were still complaining about the capacity of their group members."

Students' average grades improved compared to previous year. While we could not control for confounding factors such as differences in difficulty of the exam, ability of students, and overall teaching quality in the course, this trend implies that the students gained more knowledge and understanding of agile practices and their interdependencies.

Students' commitment to the XP practices generally improved in comparison to previous years, but differed between groups and over time. Some practices were adopted by all groups during the duration of the project, e.g., pair programming, small releases, and simple design. Others, such as Test-driven Development (TDD), few or no groups reported experiences after the first sprint. Systematically checking usage of agile practice after each sprint allowed us to react, so that seven groups reported significant experience with TDD after the second sprint. Table 3 shows the practices used over the duration of the project.

Students' work as a group was positively influenced by our changes in groups with less technically skilled students. Groups that lacked previous knowledge and experience in software development, working with repositories, etc., could focus on the development process and to some regard even surpass groups with more technically skilled students. We assume that this is due to different preferences among students. Some technically inclined students might bring little interest for process related topics and could dominate a group of less technically inclined students towards putting less emphasize on agile methods.

Consequently, the changes in the course focus had positive effects on the teaching but only had a mitigating effect on reducing the students' cognitive load.

Next Iteration. We plan the following changes for the next iteration: *Provide a framework* to students, e.g., a skeleton android project derived from previous year, to reduce technical challenges. *Intensify reflections* on agile practices in Sprint retrospectives and during sprints. *Emphasize on TDD* to further align product development and agile method conformance. Based on a pre-arranged framework (see above), this can help students to create meaningful Unit, Integration, and Acceptance tests.

Despite mixed feedback, we intend to keep *Android as a teaching platform*. Students perceived Android as difficult, but also as a valuable skill. Other frameworks (Java Swing or web-frameworks) do not promise significant improvements. We hope to address challenges of applying TDD with Android with specific infrastructure.

5. DISCUSSION

While our experiences are unique in each course, they do follow a number of common themes. We identify these themes in this section and discuss the challenges and changes, the effects and lessons learned, as well as the limitations and opportunities below.

5.1 Challenges and Changes

Babb et al. [4] report on four different areas where problems occur when learning agile in an industrial setting: multiple goals, excessive iteration pressure, customer involvement, and organisational culture. Interestingly, we find that this categorisation also applies to the challenges we identified and to the changes we made to address them.

Multiple Goals. At the beginning of the academic year, the setup of the project courses forced students to acquire domain-specific content knowledge, technological knowledge, and pedagogical knowledge (the process knowledge) at the same time. Since the level of technical knowledge is usually quite low, a lot of time is spent on GitHub, Android, etc. This is at odds with the learning objectives concerning process knowledge. Both teachers and students tended to focus mainly on technical issues instead of emphasising the necessity to follow the process and assess how well the process is observed. This was particularly evident in the project courses and to a lesser extent in the Software Processes course. The Software Architecture Project introduced another form of this; since the students need to deliver a product, documentation of the product, and a report on how they applied process knowledge, they had to split their attention between these aspects. The multiple goals have been addressed by changes to the Agile Development Processes project, where grading criteria have changed and the product is no longer in focus. This removes the pressure to deliver final software and allows students to focus on applying the process correctly. In Software Engineering Project we emphasise the importance of the process in the overall assessment.

Excessive Iteration Pressure. The pressure to deliver an increment is evident in all our project courses. Due to the multiple goals students need to address and the difficulty in knowledge acquisition, this causes problems in the early sprints that influence the quality of the final deliverable. This issue has been addressed by changes to the grading criteria in the Agile Development Processes project. In the Lego Scrum simulations, however, the excessive iteration pressure is used as a pedagogic means to force issues to occur, and in the case of the Software Engineering Project to introduce mitigations to avoid reproducing issues in the project.

Customer Involvement. All of our courses show a lack of customer involvement, projects are not driven by an external product owner. For the Software Engineering Project this will be changed for the next iteration by introducing external stakeholders. The lack of customer involvement has been addressed in the Software Architecture Project by providing a student supervisor for each group that acts as a product owner. However, the teams still define the product backlog, a task usually done by the product owner. The Lego Scrum simulations also have a dedicated product owner. Other project courses do not provide such a role, thus omitting this aspect of Scrum.

Organisational Culture. Many of the issues that we observed are based on the organisational setup of the courses. The organisational factors include the schedule of the lectures and sprints, the requirements students have to fulfil, the number, frequency, and duration of supervision sessions, and others. All courses discussed in this paper has changed in this regard, e.g., by introducing simulations/miniatures, by changing sprint lengths, changing the way students assume roles, and changing the way assessment sessions are held. Software Engineering Project also changed how technical and content supervision is carried out.

5.2 Effects and Lessons Learnt

The effects described above indicate that the changes we applied are effective in putting the focus on the process aspects that we want to emphasise in teaching agile.

When teaching agile processes or principles, there should be direct feedback on how these are applied. Such feedback is only possible if the teacher(s) observe the students apply agile practices and interact with stakeholders, e.g., the Product Owner, and with each other. Feedback allows early corrections to be made to incorrectly applied practices and thus create a process improvement learning opportunity. This guided teaching is in line with a workshop environment where the teachers act as *masters*, *coaching* the students in a setting *scaffolded* to emphasise learning Scrum — not the tools — and facilitate in the students *reflections* [7] *in-action* [24]. This direct feedback can be stifled by scalability issues. If a project is conducted with many students, then review meetings tend to focus on the product instead of the process. In some settings, it is possible to alleviate the issue by having a sufficient number of observers/supervisors available. On the other hand, by creating learning opportunities, e.g., with miniatures and simulations like the Lego Scrum simulations, students can discover necessary knowledge themselves. The teacher's role is then to encourage reflection and guide the discovery process.

Many of the changes we performed were geared towards allowing students to perform *knowledge acquisition* tasks at the beginning. Agile methods can give students tools to account for the complexity of knowledge acquisition and to plan for these tasks accordingly. In a project course this implies that the teachers' expectations w.r.t. the product at the outset of the project need to be reduced and that students must be made aware of the possibility and necessity to plan for knowledge acquisition.

Teaching agile needs to have a practical element: it must be applied to know it. However, the more technically advanced the practical element is, the more the students focus on learning the platform instead of the process. The platform becomes an impediment for learning Scrum.

Stress is another impediment, students focus on delivering before the deadline instead of using a sound process. stress creates learning opportunities in a workshop setting since close interaction with the teachers enables immediate and detailed feedback, thus facilitating what Schön defines as *reflection-in action* during the sprint retrospectives.

5.3 Limitations and Opportunities

The project-based learning approach emphasises the application of theoretical knowledge in a practical setting. The original challenges with the Software Processes course show how important it is to anchor theoretical knowledge in practice. The theoretical process knowledge was dismissed after the course finished since the students did not feel it was relevant to their education. We discussed reducing lectures with theoretical content to a minimum and move all learning to the practical realm, but realised that we cannot only teach what we can also model in a project. Removing knowledge would not only jeopardise the learning objectives but also reduce the realism of the project setting. The project setting would become too abstract and too unrealistic, and students would dismiss it. We have to maintain a balance between the difference knowledge areas.

The Lego Scrum simulation shows some important differences in the way Scrum is used. As discussed below, there is a product owner present in the simulations. More importantly, however, the different development teams create a joint product rather than an individual one. Therefore, issues such as inter-team collaboration and planning across teams become important. This makes the simulations unique in the curriculum since all projects focus on one product per team and no integration is necessary. While task estimation and breakdown are important topics in the simulation, they are not their main focus. This causes issues in the project that follows since students did not acquire the understanding and skills to apply planning poker and breakdown techniques in the project setting.

All courses show a lack of customer involvement in the sense that the projects are not driven by an external product owner. This caused the students to take ownership of the product in the Software Engineering Project. Students did not learn to react to the wishes of an external customer that drives the development of the product. While the product owner role was introduced in the Software Architecture Project through student supervisors, the level of customer involvement is still less than what would be found in a Scrum project in industry where a product owner is embedded in the development team. The Lego Scrum simulations feature a product owner, albeit a somewhat malicious one as a means to create learning opportunities. To implement full customer involvement in a teaching context is, unfortunately, prevented by resource considerations, so we must be content with a best-effort solution.

6. CONCLUSIONS AND FUTURE WORK

In this paper we reported on our ongoing discussions on how to improve the way we teach agile and how we applied our insights to four different courses. We showed different approaches to address the challenges and their effects. In particular, we used a Lego Scrum workshop to introduce agile ideas, organised student work to adhere to Scrum principles better and to give more insight into the application of the process, changed course assessment to emphasise the importance of applying the process correctly, and put more emphasis on retrospectives and process improvement. Overall, the effects of these changes were positive and seem to have improved student engagement with agile principles and practices as well as the teachers' confidence that students pick up the knowledge and skills important w.r.t. the application of agile processes. By sharing our experiences, our lessons learned, and our reflection on our progress, we provide insights that will allow teachers elsewhere to benefit from our experience and to start the same alignment process across courses oriented towards agile software development.

Our report was created in the context of the Software Engineering and Management programmes and other programmes that contain Software Engineering courses at the University of Gothenburg and Chalmers University of Technology. An important property of these programmes is their emphasis on problem-based and project-based learning. This is evident in the fact that three of the four discussed courses are project courses in which students are asked to tackle a concrete problem. While the specific environment in which we applied our changes might have an impact on their effectiveness, we believe that our findings can be generalised to the extent that they provide helpful guidelines for others or at least start a discussion about the issues reported.

In our future work, we will analyse the changes we have described here for the next iterations of our courses. We will also continue our discussions and develop new ideas and approaches to improve the learning environment in the courses addressed here. In addition, we are working on extending our methods to new courses to achieve similar effects.

Acknowledgement

Jan-Philipp Steghöfer and Håkan Burden are thankful for Dominik Haneberg's help in setting up the Lego Scrum simulation. The authors also thank the numerous students who helped us improve how we teach agile processes.

7. REFERENCES

[1] E. Alégroth, H. Burden, M. Ericsson, I. Hammouda, E. Knauss, and J.-P. Steghöfer. Teaching Scrum–What We Did, What We Will Do and What Impedes Us. In *Proc. XP 2015*, volume 212, page 361. Springer, 2015.

[2] Z. Alzamil. Towards an effective software engineering course project. In *Proc. ICSE '05*, pages 631–632. ACM, 2005.

[3] O. Astrachan, R. Duvall, and E. Wallingford. Bringing extreme programming to the classroom. In *XP Universe 2001*, 2001.

[4] J. Babb, R. Hoda, and J. Nørbjerg. Barriers to Learning in Agile Software Development Projects. In H. Baumeister and B. Weber, editors, *Agile Processes in Software Engineering and Extreme Programming*, volume 149 of *LNBIP*, pages 1–15. Springer Berlin Heidelberg, 2013.

[5] K. Beck. *Extreme Programming Explained: Embrace Change*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2000.

[6] J. Biggs and C. Tang. *Teaching for Quality Learning at University*. Open University Press, 3rd edition, 2007.

[7] J. S. Brown, A. Collins, and P. Duguid. Situated Cognition and the Culture of Learning. *Educational Researcher*, 18(1):32–42, January 1989.

[8] M. Brydon-Miller, D. Greenwood, and P. Maguire. Why action research? *Action research*, 1(1):9–28, 2003.

[9] A. Cockburn. Elephant Carpaccio Exercise. online. http://alistair.cockburn.us/Elephant+Carpaccio+Exercise. Accessed Oct. 20, 2015.

[10] M. Cohn. *Agile Estimating and Planning*. Prentice Hall PTR, 2005.

[11] R. Corey. *Case Method Teaching*. Number 9-581-058. Harvard Business School, 1998. Rev. November 6.

[12] M. Holcombe, M. Gheorghe, and F. Macias. Teaching XP for real: Some initial observations and plans. In *XP2001*, Sardinia, Italy, 2001.

[13] D. Johnson and J. Caristi. Extreme programming and the software design course. In *XP Universe 2001*, 2001.

[14] M. Koehler and P. Mishra. *The handbook of technological pedagogical content knowledge (TPCK) for educators*, chapter Introducing TPCK, pages 3–29. American Association of Colleges of Teacher Education and Rougledge, NY, New York, 2008.

[15] A. Krivitsky. LEGO Scrum Simulation. online. http://www.lego4scrum.com/. Accessed Oct. 20, 2015.

[16] D. Lübke and K. Schneider. Agile Hours: Teaching XP skills to Students and IT Professionals. In *Proc. of Product-Focussed Software Process Improvement (Profes '05)*, Oulu, Finland, 2005.

[17] P. Merel. eXtreme Hour. online, 2001. http://c2.com/cgi/wiki?ExtremeHour. Accessed Oct. 20, 2015.

[18] M. Paasivaara, V. Heikkilä, C. Lassenius, and T. Toivola. Teaching students scrum using LEGO blocks. In *Proc. ICSE '14*, pages 382–391. ACM, 2014.

[19] M. Paasivaara, C. Lassenius, D. Damian, P. Räty, and A. Schröter. Teaching Students Global Software Engineering Skills Using Distributed Scrum. In *Proc. ICSE '13*, pages 1128–1137. IEEE Press, 2013.

[20] V. Peeters and P. Van Cauwenberghe. *Extreme Programming Perspectives*, chapter The XP Game Explained, pages 311–321. XP Series. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, 2003.

[21] S. Price, Y. Rogers, M. Scaife, D. Stanton, and H. Neale. Using 'tangibles' to promote novel forms of playful learning. *Interacting with computers*, 15(2):169–185, 2003.

[22] C. Scharff, O. Gotel, and V. Kulkarni. Transitioning to Distributed Development in Students' Global Software Development Projects: The Role of Agile Methodologies and End-to-End Tooling. In *Proc. ICSEA 2010*, pages 388–394. IEEE, Aug 2010.

[23] J.-G. Schneider and L. Johnston. eXtreme Programming at Universities: An Educational Perspective. In *Proc. ICSE '03*. IEEE CS, 2003.

[24] D. A. Schön. *The Reflective Practitioner: How Professionals Think in Action*. Harper torchbooks. Basic Books, 1983.

[25] K. Stapel, D. Lübke, and E. Knauss. Best Practices in eXtreme Programming Course Design. In *Proc. ICSE '08*, pages 769–776. ACM, 2008.