From EDAF45 to EDAG05: a Course Transformation Story

Emma Söderberg
Dept. of Computer Science, LTH

Abstract

The department of computer science offers an appreciated course on software development in teams (EDAF45) that for scheduling reasons currently can't be offered beyond the D programme. To address this issue, the department is developing a new course on agile software development (EDAG05) that can be offered to non-D engineering students. In this report, we gather data about EDAF45 and identify issues that we would like to address in the new course. The result of this review is a proposed format for EDAG05 where we try to address the identified issues.

1 Introduction

The department of computer science has a course, 'Software Development in Groups' (EDAF45), teaching agile software development and targeting the D programme. EDAF45 has roughly the following structure; it runs over two study periods, during the first study period there are lectures and labs in preparation for the project, which runs in the second study period. The project is designed to simulate a software project with a team of developers (the students) and a customer (played by a teacher), and is centered around six development iterations.

EDAF45 has a couple of unusual elements; the project is organized around 6 iterations, each mapping to a week where students have a mandatory full day lab (8 hours). During the project, student groups are coached by students taking a parallel course on coaching of software teams. This special arrangement, further described in [Hedin et al., 2005], has been successful, but the setup is expensive and difficult to schedule. The scheduling issue is currently prohibiting the course from being adopted by programs beyond the D programme.

Consequently, a new course, 'Agile Software Development' (EDAG05), is being developed. Like EDAF45, the new course aims to teach students how to develop software in small teams, but in addition it should be cheaper and easier to schedule. With this in mind the goal is to design a course that runs during one study period, with less scheduled time, and with no team coaches. Given the appreciation of EDAF45, the structure of EDAF45 is being used as a starting point for the development of EDAG05.

In this report, we will analyze the format of EDAF45 (structure, content, and student assessment), analyze existing course evaluation data gathered via CEQ, and gather and analyze additional qualitative data from past and current teachers responsible for the course.

In our analysis we identify a number of issues which we address one by one and explain how we plan to address these in the EDAG05. For some of the issues, we iterate the proposed approach with our interview subjects to get their feedback.

The rest of this report is structured as follows; we start with some background in Section 2, continue with an analysis of EDAF45 in Section 3, introduce and analyze EDAG05 in Section 4, and then we summarize in Section 5.

2 Background

In this section, we give a brief description of agile software development (Section 2.1) and two pedagogical frameworks (Section 2.2 and Section 2.3), referenced later in this report.

2.1 Agile Software Development

Agile practices have become mainstream in software development. A defining moment for the use of agile methods was the conception of the Agile Manifesto [Beck et al., 2001]. The Agile Manifesto, aimed at improving software development, was formulated in 2001 and was quickly adopted by the industry to such an extent that in 2019 a world-wide annual report found 97% of businesses using agile methods [State of Agile]. In short, the agile manifesto [Beck et al., 2001] emphasises the following values; (1) individuals and interactions over processes and tools, (2) working software over comprehensive documentation, (3) customer collaboration over contract negotiation, (4) responding to change over following a plan. These core values have led to agile practices that incorporate real-time knowledge and feedback throughout the development process, leading to better product quality that better fit user needs.

Extreme Programming (XP) [Beck, 2004] is one software development practice outlined at around the same time as the Agile Manifesto. XP describes twelve practices for four software development activities (coding, testing, listening, and designing) centred around five core values (communication, simplicity, feedback, courage, and respect). The twelve practices are divided into four areas; fine-scale feedback (pair programming, planning game, test-driven development, whole team), continuous process (continuous integration, refactoring, small releases), shared understanding (coding standards, collective code ownership, simple design, system metaphor), and programmer welfare (sustainable pace).

2.2 The Achievement Unlocked Approach

The Achievement Unlocked approach has been used by [Wrigstad & Castegren, 2017] in a programming course at Uppsala University with 120-140 students. Using constructive alignment [Biggs, 1996] as an inspiration, they organise their course around a list of achievements and students can “unlock” these achievements via demonstrations during labs, assignments, and a project in the course. The idea of unlocked achievements connects to mastery learning [Bloom, 1968], where students need to achieve a level of mastery (to at least 80%) before moving on to subsequent knowledge. There is also a connection to flipped classrooms [Bergmann & Sams, 2012] in the approach, because students consume material outside the classroom and work on assignments and spend the time in the classroom on discussion.

In the course described in [Wrigstad & Castegren, 2017], there are around 70 achievements (presented in groups of connected achievements) and each achievement is connected to a grade (3-5). A student must unlock all achievements for a grade to get the grade. Students decide when they want to demonstrate achievements, and they are responsible for matching achievements that go well together. There are plenty of opportunities for demonstrations, but the number of slots are limited to around 30. The outcome of a demonstration (with typically one teaching assistant and two students) is one of three; pass, fail, or fail with pushback. Students can retry as many times as they want and during the same slot (except if they failed with pushback when they need to wait until the next slot). During a demonstration students state which achievements they wish to unlock, why/how these will be demonstrated together, and what evidence they will use in the demonstration.

2.3 The Assessment Cycle

In [Reinholz, 2016], Reinholz presents a model called *the assessment cycle*, linking peer assessment to self-assessment. Self-assessment is defined as having three core elements; *goal awareness*, *self/performance awareness*, and *gap closure*. In short, these elements capture where you want to be, where you are currently at, and how you go about closing the gap of where you are and where you want to be.

With elements of self-assessment in mind, the assessment cycle includes the following steps; task engagement (step 1), then peer analysis (step 2), then feedback provision (step 3), then feedback reception (step 4), then peer conferencing (step 5), then revision (step 6), and then back to step 1. This model extends a previous model of peer assessment with the addition of step 2 and 5. The goal of these additional steps is to capture the connection to self-assessment on a more fine-grained level. For instance, step 2 is described as connecting to goal awareness as it provides more examples and a possibility to see quality variation, while step 3 aids performance awareness both by explanation of ideas (by the student) and by the receiving of feedback on explanations (from the teacher).

3 EDAF45: An Appreciated Course

In this section, we gather data about EDAF45 and identify issues to address in EDAG05.

3.1 Method

To analyze EDAF45 we gathered data about the course from the course web page, from three past and current teachers responsible for the course, and from CEQ data gathered between 2017-2020. The data from teachers was qualitative data gathered via semi-structured interviews carried out by the author (protocol is included in Appendix A.3).

We will refer to the interviewed teachers as T1 (currently responsible for the course), T2 (currently responsible for the companion course EDAN80), and T3 (responsible in the past). All teachers have worked with the course for a long time; T1 has been responsible for the course for several years, and both T2 and T3 were part of the original development of the course. The protocol for the interviews is included in Appendix A.3. In addition to these interviews, the author also had several informal discussions with T1, who is also part of the development of

EDAG05. These informal discussions have provided continuous feedback, during the course development process, and the teacher has acted as a critical friend throughout this project.

3.2 Results

We present the result in terms of the structure of the course (Section 2.2.1), student assessment (Section 2.2.2) and course assessment (Section 2.2.3).

3.2.1 Structure

EDAF45 has been given in its current form, as a mandatory 7.5hp course to second year D programme students for roughly 20 years, and is an appreciated course that consistently gets high scores on the CEQ evaluation (Figure 2). The structure of EDAF45 and the experience of running the course in its current form has been summarised in [Hedin et al., 2005]. The course plan for EDAF45 is included in Appendix A.1. The course runs over two study periods, as illustrated in Figure 1. The bulk of lectures and labs occur in the first study period, while the second study period is centred around the project. The first study period ends with a lecture where students are tested, in preparation for the project, and the project, which is the same for all teams and prepared by the teachers, is presented. The content of the project has been the same since the project started.

In the project, students work in teams of up to 10-12 students during six development iterations. Each iteration has an exercise session for planning and a mandatory scheduled 8 hour lab for software development. In between planning and development, each team member should spend 4 hours of self-study time on a so-called *spike*. A spike is a deep dive into some area relevant to the team's work, for instance, testing out a tool, implementing a test of a functionality etc. During the project teams are coached by students from a parallel course on coaching of software development teams (EDAN80). At the end of the project there is a project event and a final lecture to summarise the course.

During project labs students are supposed to follow the practice of XP, briefly described in Section 2.1, which advocates for practices like pair programming (two students working together in front of one screen), test-driven development, simple design, and reflection. While pair programming, students sit in pairs in front of one screen, taking turns to be the driver (writing the code) and the partner (reviewing the code) while having a continuous discussion about the code (with the driver thinking aloud while writing).

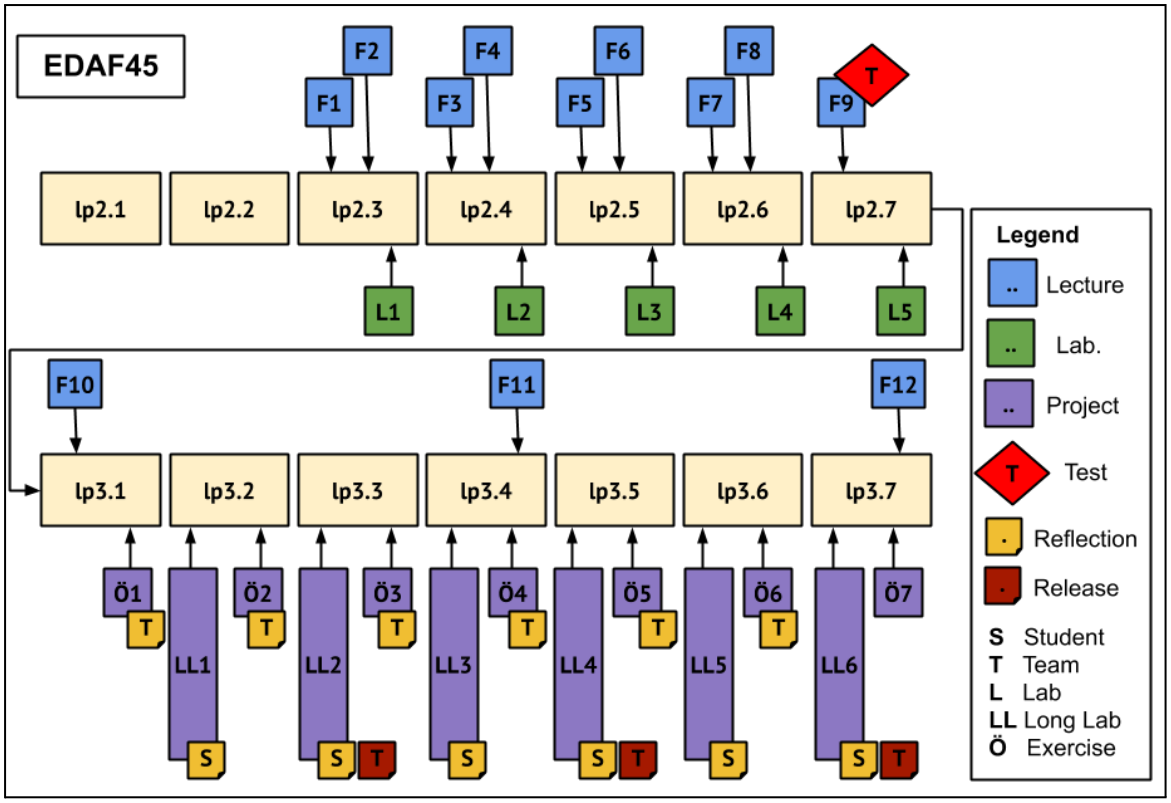

Figure 1: Course Structure Overview of EDAF45. The structure of EDAF45 as it is laid out over two study periods. The length of activities indicate their length in time, e.g., long labs are 8 hours and lectures are 2 hours. Reflections and releases are handed in.

3.2.2 Student Assessment

During the course students hand in reflections, individually and in teams. These reflections are very short, from 1-2 sentences to a paragraph. To take part in the project, students have to pass a test (kontrollskrivning). During the project, students show releases to the project customer (played by a teacher) and to other students for a final project review via peer assessment. At the end of the project, each team presents the project to another team and takes part in a project event where they demo their project. The course does not have a final examination.

3.2.3 Course Assessment

A summary from the four most recent CEQ reports for the course (2017-2020), are shown in a plot in Figure 2. Reviewing the plot, we see that the result presented for the summarising questions from the questionnaire hover around 50 and above on a scale from -100 to 100. Overall, the results over the last couple of years are fairly consistent, with some fluctuation for 'course satisfaction' and 'appropriate workload' (but these are still roughly in the range above 50). The score for 'perceived importance' stands out at consistently being close to 100.

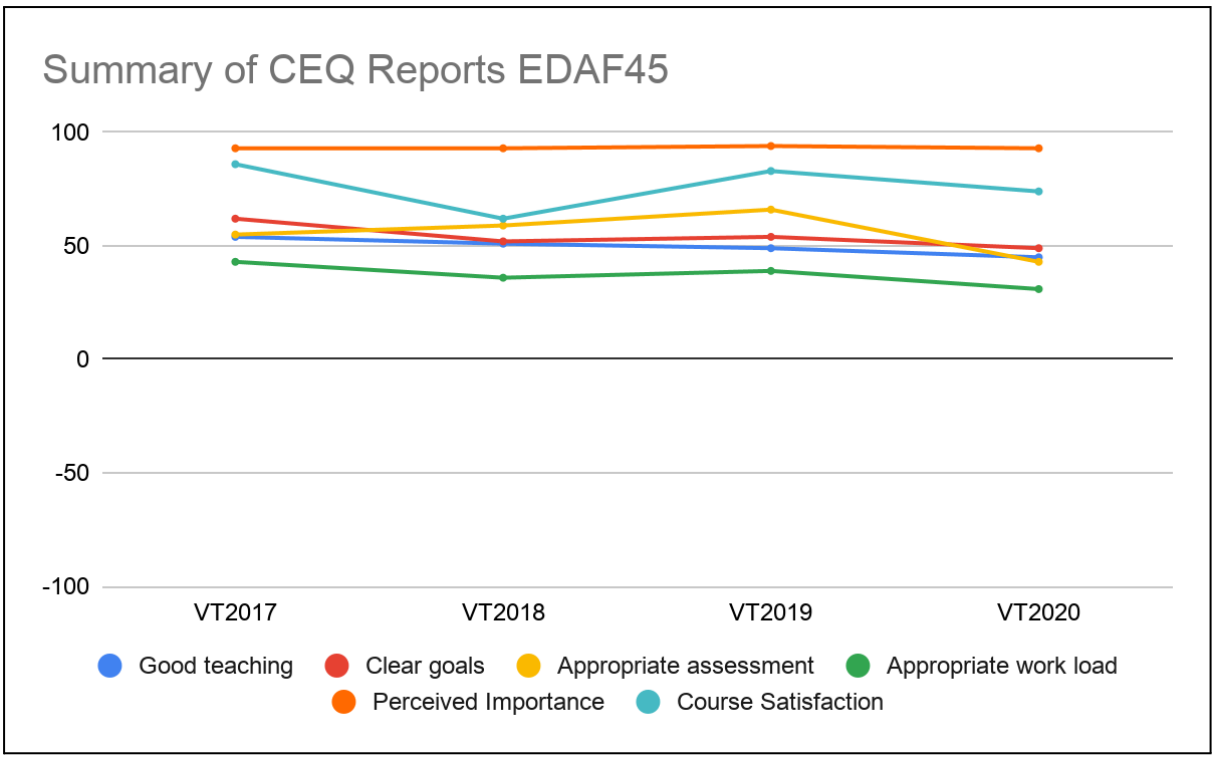

Figure 2: Summary of CEQ Reports for EDAF45 (2017-2020). The left bar shows the scale used in the CEQ form and the included results are from the summary of the questionnaire listed in the beginning of each yearly CEQ report.

To further assess the course, the author conducted semi-structured interviews with three teachers connected to EDAF45 or its companion course EDAN80.

When the teachers were asked about **what works and doesn't work in the course** (A.3. Q1) the general impression was that most things work well in the course. T1 mentioned that a pair of researchers, studying the work in teams during the last instance of the course, had observed less design discussion and distribution of work between team members than what would be preferable, which could be something to try to improve in the course. T2 mentioned that the course project is designed to be open and flexible, but one consequence of this is that teams can make their way through the whole project without having to refactor their code. T3 mentioned that students learn a lot from each other but they may learn different things depending on the students. Teams may also experience problems, due to personal differences, issues with coaches, and so on, which may give students a negative experience. T3 also mentioned that some of the practices in the course are by design "extreme", for instance, pair programming all the time is extreme but it's good for students to learn how to think out loud (which is part of the practice). T1 also mentioned some history behind the unconventionally late lectures in the structure of the course, and that the reason is to not lose too much momentum before the project.

When asked about **observed problems in teams and how to mitigate these** (A.3 Q2). T1 and T3 mentioned that there may be an imbalance between team members, for instance, a student may be dominant and not let others speak (but coaches can notice this and try to do something about it). One challenge is when students are more technically skilled than the coaches and try to pull in one direction, this can be fine (we want students to show technical leadership) but we want the whole team to go in the same direction. Also, sometimes students think they are

skilled when they are not. T2 mentioned that teams sometimes spend too much time talking about stories during planning meetings and not as much on reflection to improve the practice in the team. Another problem is that teams may start to try test-driven development (TDD) but give up as soon as it gets difficult. In both these cases coaches can help, for instance, by setting off time for reflection (mentioned by T2) or encouraging TDD (mentioned by T1). T2 mentioned that the free-rider problem is mitigated with scheduled labs and pair programming - it's hard to be an entirely passive member of a team.

Finally, when asked about **self-coaching teams** (A.3-Q3), T1 mentioned that they have run two instances of the course with self-coaching teams; the first with team members that also took part in the meeting for the coaches in the other source (did not work so well due to those meetings covering a lot of literature in the coaching course, but it was also problematic to discuss the teams while there was a team member present), and the second instance with teams coaching themselves with extra assistance from the teacher acting as customer. T2 saw problems with self-coaching in that a team member coach has less overview, and it also does not work well to rotate coaches, as there are some additional tasks that you need to do as the coach. T2 suggested having explicit tasks during an iteration (during 2h or so) to create structure that the coaches otherwise provide. The coaches often act as project leaders in the beginning, (providing structure at meetings, organising morning and lunch stand-up meetings, fika with reflection), but they hand this over to the teams as they ramp up. T1 mentioned an example of a team with refactoring growing out of proportion in a self-coaching team. T1 also mentioned feedback from students where a mix of coached and self-coached teams was perceived as unfair, especially as coaches work in pairs giving the appearance that there should be more coaches available. T3 thought self-coaching teams may have benefits as teams then would need to take responsibility and ownership of the problem faster.

3.3 Analysis

Reviewing the material gathered about the course we identify the following Issues:

- **Two Study Periods** One reason why EDAF45 is considered difficult to schedule is that it runs over two study periods.
- **Long Scheduled Labs** The full-day mandatory labs are making the course difficult to schedule for programmes other than the D programme.
- **Team Coaches Are Expensive / Too Few** Teams are coached by students from a parallel course on coaching of software development teams (EDAN80). This arrangement creates strong coupling between these courses which incurs scheduling and cost penalties. Recently there has not been enough coaches, resulting in forced self-coaching.
- **Too Focused on XP / Exposure to Few Methods** The format of the course is strongly tied to a specific method for agile software development, called Extreme Programming (XP). This method advocates for pair programming, where developers program in pairs in front of one screen, at all times. In practice, pair programming is useful for certain tasks (learning something new, debugging an issue), but not for others (mundane everyday tasks) according to [Coman et al., 2008].
- **Free Riders** Student groups are large and coaching engagement varies. Pair programming exposes programming skills and there is some pressure to participate in

this activity, but a student can still “take a back seat” and just follow along to get past the course.

- **Malfunctioning Teams** Despite coaching a team can malfunction. This can happen for many reasons but passive coaches can possibly contribute to the problem by not reacting.
- **Prioritising Progress over the Team** As mentioned in the interviews, students may focus too much on stories and not on reflection to improve the team practice.

4 EDAG05

In this section, we present the suggested format of EDAG05 and then we analyse the format by gradually stepping through the issues identified for EDAF45.

4.1 Structure

The starting point for EDAG05 is a refactored[1] version of EDAF45, squeezed into one study period, shown in Figure 3. The teams are ramped-up during an initial two weeks, then the project is run over four weeks, before the course ramps down again with one week of project presentations, demos, a guest lecture and a final lecture summarising the course. There are in total fewer lectures (six instead of nine) and labs (four instead of five). The plan is to reduce and focus the content in the lectures to what is essential for the project, and likewise for the labs. The goal is to give students all the crucial pieces of an “iteration 0” before the project, step-by-step via the labs. The number of hours for project iterations is the same, each iteration is extended to twelve hours (from eight) and the number of iterations are reduced to four (from six).

[1] Refactoring is a term used in software development. A code refactoring is a transformation of code which doesn’t change its external behaviour.

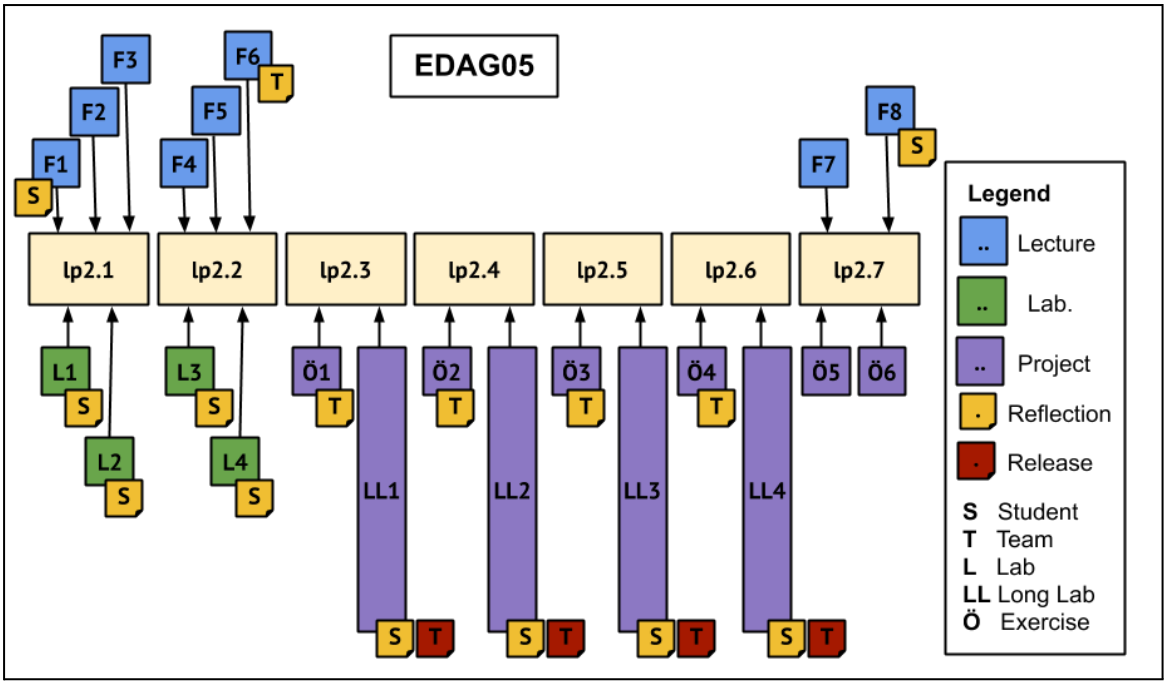

Figure 3: Course Overview for EDAG05. The planned structure of EDAG05 laid out over one study period. The size of activities indicate their length in time, e.g., lectures are 2 hours and long labs are 12 hours. Reflections and releases are handed in.

4.1.1 Teams

Teams are **self-coached and self-organised**, meaning that they define how they want to coordinate their coaching (which student will have this role; one, several, a rotation) and when they will work during each iteration. The team plans how they will schedule their iterations but will be given a framing of three days (Wed - Fri) where they should schedule a total of twelve hours for joint development work. Teams will be recommended to not schedule all the hours of an iteration in one go and to schedule at least three hours in one session. The roles in the team, expectations on teammates, and the team schedule is put down in a team contract and handed in before the project. Students taking on the coaching role will use some of their spike time (individual work done before an iteration) for this task.

To communicate what they have agreed to amongst themselves in the team, each team hands in a **team contract** outlining these details before the project starts. The use of a social contract is a practice used by others transferring agile practices into the classroom [Krehbiel, 2017], where such a contract helps to define the rules for the class.

4.1.2 Agile Methods

During the project the teams will work on a tool-oriented and representative software developer stack with tool support for quality control via code review, continuous integration and deployment, issue tracking, and planning. This tool stack and agile practices will be introduced in the labs to prepare students for the project, bringing them to a point where they have an “iteration zero” of the project. The strong coupling to XP in EDAF45 is loosened up to introduce other agile software development practices common in the industry, such as code review and distributed teams. With the use of code review students continuously peer assess

each other's work, and they can also have asynchronous digital conversations about code allowing them to work in a more distributed fashion.

Still, despite introducing code review, we want teams to learn the practice of pair programming to be able to make use of the practice when appropriate. The approach taken, to broaden the scope of the methods introduced, is to add a methods focus to each iteration: (iteration 1) co-located team and strict pair-programming (like in EDAF45), (iteration 2) co-located team working together but not using pair programming, (iteration 3) distributed team with all member sitting in a different physical location, and (iteration 4) a final mixed-method iteration where teams select what they want. With pair programming being observed as more suitable for problem solving and learning new things [Coman et al., 2008], it appears that it would be especially useful in the first and last project iterations. Having multiple methods will allow students to experience what they prefer and give them the possibility to pick the method most suitable to the task at hand.

4.2 Student Assessment

Students are assessed via achievements, reflections, and project deliverables, further explained below.

4.2.1 Achievements

During the course students and teams should unlock a number of achievements, in a fashion similar to the 'achievement unlocked' approach described in Section 2.2. An achievement is typically something that all students or teams should do to pass the course, for instance, a student should pair program with all team members, and a team should use test-driven development. An achievement is unlocked via evidence and a description of why this evidence shows that the achievement has been unlocked. The evidence should be some artefact tracked in the developer stack, for instance, a commit, a code review, or similar, where authors and activities are tracked. Students and teams decide when they want to unlock achievements and what evidence they want to use to do it.

The notion of achievements in EDAG05 is perhaps the largest change to the format of EDAF45. To test the view on using achievements in the course, a final question about achievements was added to the interviews (Appendix A.3). The general response was positive among the interviewees. T1, who also had informally discussed the potential for students to be overwhelmed with the author, suggested that achievements could be displayed on a planning board similar to how project stories are managed in the teams, making them into "stories for the course". T3 mentioned that it would be good for teams to focus on unlocking achievements rather than on finishing stories. T3 further suggested that we have more achievements than required to motivate students and to increase engagement.

With this feedback in mind, we will introduce **achievements as stories for the course** in which students need to unlock a certain number to pass. We aim to include more achievements than need to be unlocked while not making the list too long. Also, as the concept of achievements may be new to students, they will get to practise how achievements work during the labs in the first two weeks so that they are familiar with the concept when the project starts.

4.2.2 Reflections

Students and teams hand in **reflections**, where they reflect on, expectations on the course, labs, project iterations, and the final project week. A student reflection is focused on the work of the student, while the team reflections focus on the work of the team. In a reflection a student or a team, write up to a paragraph on a predefined topic, and in addition they list achievements they want to unlock. At the end of the course, students summarise their work in the course with a final reflection where they connect their work in the course with their initial expectations on the course.

The use of reflections is extensive in EDAF45, as illustrated in Figure 2, where reflections are part of the agile practices being taught in the course. For EDAG05, we add additional reflections in the beginning of the course, for the labs, and at the end of the course, to encourage students to further reflect on their learning in the course. This strategy of transferring agile practices to the classroom is inspired by [Krehbiel, 2017], reporting on successful use of reflection in the classroom.

4.2.3 Project Delivery

Similar to EDAF45, at the end of the course project each team packs up their project and hands it over to another team for review. The other team presents and demos the project, and provides peer feedback to the team owning the project. As a final event in the course, all teams demonstrate their project during a joint session.

4.3 Course Deployment

With more tool-based practices more activities become traced, e.g., commits in the version control system, comments from code reviews, issues etc. Tracing of these activities give insights into the work of a team. With this information, we can gather data that coaches typically otherwise observe in a team, for instance, the most active team member in a team (in terms of traced activities), or when the members of a team work. We can use this information to compensate for the lack of coaches and **monitor the work in the teams from a distance**. If we see “red flags”, for instance, teams not working during their scheduled time (according to the team contract), or teams primarily working in the middle of the night (not sustainable), we can react and have a discussion with the team - a kind of **on-demand coaching**.

4.4 Course Evaluation

The **final individual reflections**, summarising the experience of the course and tying them back to the expectations in the beginning of the course, will provide some evaluation of the course. Still, the primary goal is not to gather feedback about the course with this exercise but to encourage self-assessment among students [Reinholz, 2016], and in addition, the format may prevent students from giving feedback freely. To further let students give feedback anonymously, there will be a **voluntary online CEQ** after the course.

4.5 Analysis

With the suggested format for EDAG05 presented we now walk through the issues identified for EDAF45 and analyse how well these are addressed and discuss potential new risks.

4.2.1 Issue: Two Study Periods

In the new structure, the course is squeezed into one study period instead of two, making it easier to schedule. The immediate risk with this new structure is that there won't be enough time for students to ramp up for the project. This structure is more intense than that used in EDAF45, but as mentioned in the interviews there has been some concern about losing momentum in that course when it is laid out over two study periods. On the other hand, one potential benefit with this structure is that it can build momentum for the project.

4.2.2 Issue: Long Scheduled Labs

In the new format, teams schedule their own lab time, allowing each team to find common slots in their schedules. There are some restrictions in that teams need to schedule their time during three days, but the selected work sessions for an iteration may differ between teams. The use of team contracts is intended to communicate when each team will work. Still, one risk with the variation between teams is that it will be difficult for teachers to keep track of the work in the teams, with the benefit that the variation is central to making the course easier to schedule.

4.2.3 Issue: Team Coaches Are Expensive / Too Few

The use of self-coaching teams reduces the cost of the course, provided that there is a format for self-coaching that works, or the teachers in the course will end up coaching all teams. One risk here is that teams become confused and can't decide which strategy for self-coaching they want to use. This needs to be mitigated in the instructions given to teams during the ramp-up weeks in the beginning of the course. The use of a team contract has the potential of helping with clarification around the self-coaching, it at least provides a forum where the rules of the team can be discussed.

4.2.4 Issue: Too Much Focus on XP / Exposure to Few Methods

A central difference to EDAF45 is the loose coupling to XP and the introduction of modern code review, where developers use tools to review each other's changes for quality assurance before they are committed to the code base, an activity used extensively by companies such as Google [Sadowski et al., 2018]. Code reviewing and pair programming both serve a similar purpose, but they still co-exist in the industry by introduction of, for instance, a PAIR tag in commit messages for changes devised via pair programming.

From a pedagogical perspective, both of these code review practices map well to the assessment cycle described in Section 2.3, suggesting that they help students with goal and performance awareness. As an example, code review typically includes the following steps: a developer authors and uploads a change for review (similar to step 1 in the model), then assigned reviewers review the change and adds comments (similar to steps 2 and 3), then author reviews comments and potentially responds (similar to step 4 and 5), then the author

updates the change under review (similar to step 6), and then the process starts over again. Peer conferencing (step 5 in the model) is quite common in the code review process, via code review tools and face-to-face when needed. This mapping also holds for pair programming. As pointed out by T3 in the interviews (Section 3.2.3), students learn a lot from each other which is an observation that can be seen as an indication of increased goal and performance awareness due to these practices, or at least pair programming up until now.

One risk with the introduction of more agile practices is that students have to keep track of more things, but on the other hand the practices taught in the course are closer to those used in industry.

4.2.5 Issue: Free Riders

The new structure lets teams self-schedule their project iterations. The immediate risk is less oversight; teams may not work the expected hours during the project iterations. Still, some oversight can be gotten from monitoring the work in the team by observing traced activities in the developer stack. This traced information can be passed on to the teams to make it clear that there is still some oversight. In addition, the team contracts should help to clarify what the expectations on each team member are. Finally, the use of achievements will assess students individually, which means more individual assessment than in the previous course despite the reduced oversight.

3.2.6 Issue: Malfunctioning Teams

The use of monitoring to observe the work in teams should assist with detection of some red flags, like an inactive member of a team, but will be a bad tool for detection of other less obviously tracked things, such as unhealthy undercurrents in a team. As a complement to monitoring via tools, teams will interact with teachers acting as customers to the teams. During these meetings there are some additional possibilities to observe if a team is having issues.

3.2.7 Issue: Progress over the Team

As mentioned in the interviews by T3, some teams may focus more on finishing stories than the practice of the team. One potential benefit of introducing achievements is that it will shift the focus of the teams, to unlock achievements rather than finishing stories.

5. Summary

In this report, we have reviewed the format of the current course on software development in teams (EDAF45) at the department of computer science. Based on identified issues in EDAF45, a format for a new course on agile software development (EDAG05) is proposed and reviewed in light of the issues identified for the earlier course. The proposed format has a couple of major differences to the previous course in the updated course content, the more intense structure during one study period, and the use of achievements. The main risks with these larger changes is that it will be intense and that there will be too many things to keep track of for students.

References

[Beck, 2004] Beck, K. (2004). *Extreme Programming Explained: Embrace Change*. 2nd Edition. Addison-Wesley. ISBN-13: 978-0-321-27865-4.

[Beck et al., 2001] Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., Kern, J., Marick, B., Martin, R., Mellor, S., Schwaber, K., Sutherland, J., & Thomas, D. (2001). *Manifesto for agile software development*. Agile Alliance. Available at www.agilemanifesto.org.

[Bergmann & Sams, 2012] Bergmann J., and Sams, A. (2012). *Flip Your Classroom: Reach Every Student in Every Class Every Day*. International Society for Technology in Education.

[Biggs, 1996] Biggs, J. (1996). *Enhancing Teaching Through Constructive Alignment*. Higher Education, Vol 32(3): 347–364.

[Bloom, 1968] Bloom, B. S. (1968). *Learning for Mastery. Instruction and Curriculum. Regional Education Laboratory for the Carolinas and Virginia, Topical Papers and Reprints, Number 1*. Evaluation Comment Vol 1(2).

[Coman et al., 2008] Coman, I. D., Sillitti, A., and Succi, G. (2008). *Investigating the Usefulness of Pair-Programming in a Mature Agile Team*. XP'08: Proceedings of the International Conference on Agile Processes and Extreme Programming in Software Engineering.

[Hedin et al., 2005] Hedin, G., Bendix, L., and Magnusson, B. (2005). *Teaching Extreme Programming to Large Groups of Students*. Journal of Software and Systems, Vol 74(2): 133--146, Elsevier.

[Hedin et al., 2006] Hedin, G., Bendix, L., Magnusson, B., and Ohlsson, L. (2006). *Tandemo Courses . Students Coaching Students*. In proceedings of the Pedagogical Inspiration Conference 2006, Faculty of Engineering, Lund University, Sweden.

[Krehbiel et al., 2017] Krehbiel, T. C., Salzarulo, P. A., Cosmah, M. L., Forren, J., Gannod, G., Havelka, D., Hulshult, A. R., and Merhout, J. (2017). *Agile Manifesto for Teaching and Learning*. Journal of Effective Teaching, 17(2):90--111.

[Reinholz, 2016] Reinholz, D. (2016). *The Assessment Cycle: a Model for Learning Through Peer Assessment*. Journal of Assessment & Evaluation in Higher Education, Vol. 41(2): 301--315, Routledge.

[Sadowski et al., 2018] Sadowski, C., Söderberg, E., Church, L., Sipko, M., and Bacchelli, A. (2018). *Modern Code Review: a Case Study at Google*. ICSE-SEIP'18.

[State of Agile] CollabNet VersionOne (2019). *13th Annual State Of Agile Report*. www.stateofagile.com, visited May 2020.

[Wrigstad & Castegren, 2017] Wrigstad, T., and Castegren, E. (2017). *Mastery Learning-Like Teaching with Achievements*. SPLASH-E'17: Proceedings of the ACM SIGPLAN conference on Systems, Programming, Languages and Applications: Software for Humanity - Educational Track.

Appendix

A.1 Course Plan for EDAF45

Aim: *"The aim with this course is to give knowledge about and practical experience on cooperation within a software development team. Focus is on the method Extreme Programming, which uses a highly iterative work process. The course covers principles for customer cooperation, planning, sustainable design and implementation, testing, and delivery of the product. The course also gives additional training in object-oriented programming. The course also covers an introduction to program development methodology in general and the terminology used."*

Learning Goals - *Knowledge and Understanding:*

- ***LG1****: "be able to present and motivate the different practices within extreme programming."*
- ***LG2****: "be able to present principles for version control."*
- ***LG3****: "be able to define basic terms and definitions in software engineering."*
- ***LG4****: "be able to describe the most common software development processes."*

Learning Goals - *Skills and Abilities*:

- ***LG5****: "be able to develop and deliver a sustainable software product in cooperation with others."*
- ***LG6****: "be able to apply practices and tools for automated testing, refactoring, and version control."*
- ***LG7****: "be able to apply iterative planning."*
- ***LG8****: "be able to apply pair programming."*
- ***LG9****: "be able to reflect on your own and the team's activities during a development project and understand how these contribute to a successful development process."*

A.2 Course Plan for EDAG05

Aim: "*To give knowledge and practical experience of how to develop software together in a team. [The course focuses] on the practical experience of methods and tools suitable for a smaller software project with one developer team."*

Learning Goals - *Knowledge and Understanding:*

- **LG1:** *"be able to define basic concepts within software development.*
- **LG2:** *"be able to describe and motivate different techniques used in software development."*

Learning Goals - *Skills and Abilities*:

- **LG3:** *"be able to develop and deliver software in collaboration with others.*
- **LG4:** *"be able to apply techniques and tools for software development."*

Learning Goals - *Estimation and Assessment:*

- **LG5:** *"be able to assess how activities in a software project affect the development process."*
- **LG6:** *"be able to see connections between activities in the development process and the final software product."*

A.3 Protocol for Semi-structured Interviews

Questions:

1. **What works in EDAF45? What doesn't work?**
2. **What problems do teams have? Mitigation?**
3. [*After explaining that EDAF45 has been starting to use self-coaching teams*]
   How well does self-coaching of teams work? Problems? Mitigation?
4. [*After explaining the Achievement Unlocked Model*]
   What do you think about using achievements in EDAF45?