Preprint:[1]

Agile Iteration Reviews in a Project Course: A key to Improved Feedback and Assessment Practice

Torgeir Dingsøyr
Department of Computer Science
Norwegian University of Science
and Technology, Trondheim

***Abstract*—Agile development is increasingly taught at universities worldwide. Project courses are redesigned in order to better fit these methods, both with respect to content taught and how courses are organised. This position paper builds on experience with reviews as a feedback practice in a bachelor level project course. Reviews are a key element in agile development, but there has been little discussion in the software engineering education literature on the role of such reviews in improving feedback and assessment. Through examples of course improvement work in a course with about 140 students in 24 teams, we show how review practices are tailored to better comply with principles of good feedback practice which intend to empower students and help self-regulate learning. We argue that reviews can provide formative assessment and improve the learning outcome. Finally, we conclude with five lessons learned from three years of continuous improvement.**

Keywords—software engineering education, capstone course, agile software development, Scrum, self-regulated learning.

1. INTRODUCTION

Agile development is increasingly taught at universities worldwide in order to provide motivation through real-world practices, and through software development methods that focus much more on feedback than traditional approaches [1]. Many environments have reported experience with redesigning existing courses from traditional methods focusing on phase models for project management with agile methods focusing on evolutionary delivery of software. For the software industry, the change from traditional to agile methods have meant significant changes in work practices for development such as pair programming and test-driven development but also changes in how requirements are defined and managed, how and when architectural decisions are taken, how improvement work is organised, a much stronger emphasis on teamwork and other approaches to managing knowledge within and between teams [2, 3].

The traditional view of software development assumes that high quality software can be built by meticulous and extensive planning, while agile methods assume such software can be built iteratively with rapid feedback and that what is to be built is subject to change [2]. We can view this change as a move from a tradition focusing on knowledge as something explicit and fairly stable to a focus on knowledge as tacit and something subject to change within a sociocultural practice. This change of perspective will then influence how students are assessed [4]. Adopting agile methods at universities is not just a matter of changing curriculum in courses, but requires us to rethink not only how students are assessed but also how courses are structured [1].

[1] Preprint of article accepted for publication at the Software Engineering Education for the Next Generation Workshop, Joint track on Software Engineering Education and Training, *International Conference on Software Engineering* 2021.

The software engineering education literature includes a growing literature on project courses making use of agile development methods. Schneider et al. [5] review literature on capstone courses and report that a number of agile methods such as Extreme programming, Feature-driven development, Dynamic Systems Development Method and Scrum have been introduced at universities.

An example is the semi-capstone course at Purdue Polytechnic Institute where a course was reorganised to provide an "authentic learning experience" [6] which was student-centred and "delivered via active pedagogies", and "situated in a meaningful context". Iterations was used to provide student teams with formative feedback which was combined with traditional assessment methods.

That students greatly appreciate frequent feedback was described as a lesson learned from a software engineering group project at Imperial College [7], which shows trials to simulate industrial conditions with feedback given to groups at weekly intervals. Groups submitted code and also held bi-weekly demonstrations of the software product.

This experience report focuses on three years of improvement work in a bachelor programming project course where students develop a software product as a team over 14 weeks. Students are expected to demonstrate capabilities in programming from a previous course, but learn about teamwork and development process. The course had about 140 students in 24 teams who all developed the same software product. The average team size was seven members.

We present a number of changes in the course, but focus mainly on how iteration reviews were used to improve learning in a course with scarce resources. This paper is structured as follows: We first provide context by describing the course learning objectives and resources available, as well as changes over a period of three years. Then, we provide background on how iteration reviews are described in the practitioner literature on agile development, we present a framework for feedback practice with seven principles [8], and then use these to discuss the reviews as a feedback practice. Finally, we summarise main lessons learned and present future plans for continuous improvement.

2. A PROJECT COURSE WITH FOCUS ON TEAMWORK AND PROGRAMMING

The project course is described as requiring students to "*undertake a medium-scale programming project in a team of 5 to 9 students. The main aim is to give the students an understanding of the relations between the product-oriented and the process-oriented issues and work tasks in a programming project. ... Students will have to work and reflect about integration of different components to ensemble a larger software product. ... the aim of this course is to improve students' practical skills in teamwork as well as programming*". The course is taken by students at a bachelor program in informatics at the start of year two, after they have completed a course in object-oriented programming. We describe this course as semi-capstone as students are to use knowledge from the programming course to develop a larger product as a team. The resources for the course were one adjunct professor (180 hours), two teaching assistants (120 hours) and four student assistants (100 hours). The physical resources available was an auditorium for lectures and computer rooms shared with other courses for programming work. Course changes have been discussed with "reference groups" who are 4-7 volunteers who are recruited at the start of the semester and have met and discussed the course three times during the semester.

The course has gradually been changed. An earlier version included lectures on project management, software development method, an introduction to software architecture and database design. Project work started in parallel with lectures over the first five weeks of the semester. Teams were free to choose a technology and development process, but all teams were asked to develop the same product which was described in a requirements specification document. Teams were assigned an assistant who could meet weekly. Students were mainly evaluated on a written team report delivered at the end of the project which described the product developed as well as work process. In addition, students wrote an individual reflection report about the work which counted 20% of the grade. The following changes were introduced:

- *Method-focus in course:* From presenting several approaches to development method to requiring teams to use Scrum, but let them tailor the method to own needs.
- *Curriculum:* From using several textbooks to relying mainly on easy to read sources focusing on Scrum [9], retrospectives [10] and experience with Scrum [11].

- *Supervision of teams:* Teaching and student assistants played the role of *product owners* in addition to supporting teams with other topics such as teamwork challenges and technical challenges with the product under development. Weekly meetings with assistants, and a weekly coordination meeting amongst assistants. All meetings conducted physically in an auditorium during four hours (six teams in parallel).
- *Assistant training:* We introduced a two-day training of assistants before the start of the course to give assistants a more thorough background on agile software development and Scrum. Further, we held several workshops to improve feedback to students and conducted regular retrospectives with assistants to capture ideas for course improvements.
- *Lectures:* Instead of having 2-hour lectures over the first five weeks of the course, we conducted all training on project management and Scrum during the first week (4 hours), and then focused on setting up the development environment and Github in week 2 (4 hours).
- *Product source code:* This was delivered together with project report at the end of the course, we required all teams to commit code in a common repository where the assistants could follow progress. We also conducted analyses on number of active code contributors in each team.
- *Team report:* We put a larger emphasis on reflections on own experience in using the development method, and also asked for technical documentation of the product.
- *Individual report:* We changed the focus from individual experience with method to selecting one of five key topics for individual reflection: Daily meetings, Teamwork, Retrospectives, Release planning or Estimation.
- *Portfolio assessment:* We let reviews count 30% of the evaluation, the team report 40% and an individual report 30%.

3. Iteration Reviews in Agile Development

Reviews are a key practice in the Scrum development method, and also an opportunity for feedback in a project course. The practitioner literature describes iteration reviews as a meeting to inspect a product increment, which is an "*informal meeting, not a status meeting and the presentation of the increment is intended to elicit feedback and foster collaboration*" [12]. The team is advised to focus on what has been *done*, show the actual working code, keep focus on a "business-level" such as on user stories, and not spend much time preparing for the review.

Kniberg [9] lists a number of benefits with review meetings such as i) the team gets credit for what they have accomplished, ii) other stakeholders learn about the product, iii) the team gets feedback from stakeholders, iv) it is a social event where teams can interact and discuss their work, v) it puts emphasis on completing work, making sure that new features are "really done". Rising and Janoff´s experience report [11] describes Scrum as a method which develops the relationship with the customer, builds trust, grows knowledge and creates a culture where everyone "expects the project to succeed".

4. Feedback Practice for Self-Regulated Learning

Feedback serves a number of purposes in teaching. Price et al. [13] critically discuss feedback practice and identify five roles of feedback: Providing correction of student misconceptions, provide a positive or negative reinforcement of behaviour, forensic diagnosis, benchmarking and longitudinal development. The role that feedback has will also be influenced by when the feedback is given, how it is given, and by who. An empirical study on feedback [13] found that students in general were critical to feedback received, which was perceived as having an “overly negative tone” or was described as “vague” or “ambiguous”. Teachers recognised the importance of feedback but had few ideas of the effect of the feedback apart from seeming to think that a large volume of feedback lead to increased learning.

In a widely used article on formative assessment and self-regulated learning, Nicol and Macfarlane-Dick [8] describe seven principles for supporting and developing learner self-regulation:

1. *Clarify what good performance is* - goals, criteria and standards, can be clarified in written statements with assessment criteria.

2. *Facilitate self*-assessment - create structured opportunities for self-monitoring and the judging of progression towards goals.
3. *Deliver high quality feedback information* - relate feedback to goals, standards or criteria.
4. *Encourage teacher and peer dialogue* - increase understanding of feedback by allowing students to correct misunderstandings.
5. *Encourage positive motivation and self-esteem* - use low-stakes assessment tasks with feedback geared to provide information about progress and achievement.
6. *Provide opportunities to close the gap* - provide opportunities for resubmission of assignments to close gap between current and desired performance.
7. *Use feedback to improve teaching* - generate cumulative information about students' understanding and skills.

In the following, we use these principles in discussing reviews in the project course as a feedback practice. Note that we did not use these principles in designing changes, but in the retrospective analysis of the changes.

5. Reviews as a Feedback Practice

We wanted to use reviews for two purposes, as a feedback practice, and as project milestones. As a feedback practice, reviews are done often and could both provide students with an indication of progression towards learning goals (principle 2), be a low-stakes assessment task in that they count only 10% of the grade (principle 5) and that they are conducted regularly would make teams able to close gaps between current and desired performance (principle 6). As milestones, the reviews encourage students to get started with work in the course early, and also ensure that the students actually spend time developing a product (and not only focus on the team- and individual report), so they get experience with development method during the course and have experience to reflect on in summative team- and individual reports.

The first review was already in week 4, but was a presentation on method choices, an initial test plan, release plan and sprint backlog for first iteration rather than a proper product review. The following two reviews were of the real product, where the team got five (review2) and seven (review3) minutes to show functionality. The reason for the short time was that we wanted to let all teams demonstrate during one full day and had limited resources for assessment. Also, we did not want to make the reviews a "big thing", each review counted 10% on the final grade, but assistants were asked to downplay the role of review. We recommended the following agenda:

1. Show the release plan for your product
2. Show the most important user stories implemented, show that the user stories satisfy test conditions

To emphasise that the work is conducted in teams, we asked the whole team to be present. One team member presented, and the rest of the team could engage in answering questions after the presentation. We showed time remaining on a laptop, while the team could use one screen to show product and another screen to show the release plan of the product to make it easier to follow. From the teaching staff, a teaching assistant and a student assistant followed the review. The evaluation criteria were:

- Ability to show status on product
- Ability to show how the product satisfies test conditions

We emphasised that ability to present was not part of the evaluation criteria. The grades on reviews gradually got better as teams began working together and understood better how to prepare.

Critique after the first year was that the teams wanted more feedback from reviews than a quick oral feedback after the review and a grade (principle 3). Also, we were unsure if the teams prepared for the reviews as a team or if this work was assigned to one or two members and the rest of the team would just join the meeting. This led to two improvement actions:

TABLE I. DIMENSIONS IN RUBRIC USED IN REVIEWS

Dimension	*Low*	*Medium*	*High*	*Feedback*
Show status of product				
Show that main user stories are implemented				
Only show completed user stories				
Show that user stories satisfy test conditions				
Show how the product handles errors				
Show a realistic test scenario				
Correct use of terms and methods				
Reasoning and reflection				
Total impression				

First, we developed an *analytic rubric* [14] which assistants completed during reviews, finalised in a meeting shortly after all reviews had been conducted, and gave back to teams on paper in their weekly supervision meeting the following week.

The rubric was developed through several workshops with assistants who would use them. Dimensions are shown in Table 1 and the form also included a "total impression". Teams were scored "low", "medium" or "high" on each dimension and also got qualitative feedback. For example, in the first review teams got feedback from "you spent much time on describing topics not asked for" to feedback directly connected to learning objectives such as "when you describe user stories for your product you should avoid describing how it should be implemented such as when you describe what will be in a database". At the last review, teams got feedback such as "are these the most important user stories?", and "the product backlog includes technical tasks".

The rubric was given back to the team on paper at the meeting with the student assistant the week following the review. The assistant gave an oral explanation for the score that each team got.

Second, to make sure that the learning effect of reviews was for the whole team and not only for some team members, we tried out a method which has been found to increase learning effect. Instead of letting the team decide who was presenting, we asked everyone to be prepared to present and then *randomly selected one team member for oral presentation*. All team members could help to answer questions afterwards. The reference group was sceptical to this change at first, but agreed to try it out. In practice, we started the review by lining up all team members, and then ran a wheel of fortune which selected which member was to present. The feedback from the reference group was that students felt they learned more from this approach. However, the survey to all students after course completion showed that the self-evaluation of learning effect of reviews did not change – reviews were still considered to have a smaller learning effect than writing the team report (and a larger learning effect than writing the individual report). One could think that preparing for the review as a team would have an effect on poorer students, but when examining grades given there is no significant difference after introducing this change. However, it is difficult to compare between years as the reports will be different as teams are developing another product and team composition will influence choices on method.

Despite of limited resources, the students gave feedback in questionnaires indicating that the main learning objective was reached – only 6 of 185 respondents (over three years) “disagree” that they got a better understanding of the relationship between development process and software product. Qualitative

feedback included statements on good learning outcome and organisation of course, such as "*Such a great course to get a better understanding of the process of a software development project. Great to work with a good group of people and having meetings with a 'customer' every week makes it feel more real and exciting*.", "*It is a very good structure on the course, and most work well. It is fun to develop a product and you learn a lot from the experience*." and "*Good teaching assistants, good curriculum, learned a lot through the project*". Suggestions for improvement regarding user stories, weighting of product versus reports, team size as well as the evaluation criteria: "*More thought-through user stories, especially the ones announced during the course. They seemed not to be well connected to what the product should offer.",* and *"Evaluation criteria needs to be better specified. ... Very little information was given about the different deliverables."*

Further, we asked students about the perceived learning effect of deliverables. They value the team report as the deliverable leading to most learning, while reviews have a slightly higher average than individual reports. Most agree that there is a learning effect in all deliverables.

The portfolio assessment leads to more evaluation work, but the current set-up achieves a number of purposes: It encourages work during the semester and early initiation of teamwork with reviews, it encourages team collaboration with making the product and team report, and with an individual report counting 30%, about 25% of students change their grade compared to their team, which means that you are not fully dependent on being in a "good" team to get a reasonably good grade.

In future courses, we would particularly like to work further on clarifying evaluation criteria (principle 1) – which can be challenging in a project course where teams have different prior skills and experience different challenges during the project work, and in introducing practices to facilitate self-assessment such as involving teams in giving feedback to other teams during review meetings or provide feedback on draft team or individual reports.

6. LESSONS LEARNED

This experience report summarises improvement work when redesigning a project course for second year bachelor students, with about 140 students organised into 24 development teams, making a software product over 14 weeks. We have described the course organisation of a course which has been perceived as relevant and motivating by students. All course deliverables are perceived by students to have a learning effect. The reviews are a key practice in agile software development methods, and we have sought to use these meetings as a feedback practice, as a project milestone and as a part of formative assessment of the teams.

Although there are many effects of reviews in the practitioner literature which was not replicated in our course setting, as we for example did not have a real product owner and real stakeholders for the product developed, we believe the reviews had a function in giving credit to the teams for their effort in product development, it was further a social event and a clear milestone for the teams.

In this experience report, we used seven principles for supporting and developing learner self-regulation in discussing use of reviews. We hope the description of the course improvement can inspire others, and we would in particular like to emphasise the following lessons learned:

1. Introducing reviews enables *formative assessment* into a project course which traditionally relied on summative assessment. Keeping review meeting short and a lean process for providing feedback ensures a good learning outcome with scarce resources.
2. The *evaluation criteria* provided a starting point on conceptualising what a good performance was when student teams conducted review meetings.
3. Using *analytic rubrics* after review meetings provided teams with feedback on strengths and weaknesses, the rubric developed can be useful for other courses.
4. The *assistant training* was essential in giving assistants sufficient background to provide feedback, and a broader background for providing qualitative feedback on the relevant learning objectives through the rubrics.

5. The choice to *randomly select a presenter* ensured that the whole team needed to learn and underlined a key principle in agile development that the team is collectively responsible for what is produced.

Relatively frequent, short review meetings with quick feedback based on a set of criteria linked to main learning goals have led to a quick start of teamwork and performance improvements over time. Although students indicate they learn more from the team report, the learning effect is clear and the resources needed for conducting the reviews are moderate. Future improvements to the course will involve clarifying learning objectives and evaluation criteria.

AUTHOR PROFILE

Torgeir Dingsøyr is professor in software engineering – agile at the Department of Computer Science, Norwegian University of Science and Technology. He is further adjunct chief researcher at the SimulaMet research laboratory. His research has focused on teamwork and learning in software development, as well as development methods for large software projects and programs. He has published in the software engineering, information systems and project management fields.

In 2019 he was awarded the Project Management Institute "paper of the year" award for an article on coordination of knowledge work in larger-scale agile development, with co-researchers Nils Brede Moe and Eva Amdahl Seim. With Nils Brede Moe, he was awarded the SINTEF prize for outstanding research in 2020.

He co-edited the book "Agile Software Development: Current Research and Future Directions" (2010), co-edited the special issue on Agile Methods the Journal of Systems and Software (2012), the special section on continuous value delivery in Information and Software Technology (2016) and the special section on Large-Scale Agile Development in IEEE Software (2019). He has further been program co-chair for the Evaluation and Assessment in Software Engineering (EASE) Conference (2019) and for the International Conference on Agile Software Development (2015).

He has a broad teaching experience from courses at the Norwegian University of Science and Technology where he has had one course per year from 2006, at bachelor, master and PhD level. He has further taught a course on agile project management at the Kristiania University College in Oslo (master level) and a course on Introduction to Large-Scale Agile Software Development at the Technical University of Munich (master level). He is currently co-teacher of the bachelor level software engineering course at the Norwegian University of Science and Technology which will have about 500 students in the spring of 2021.

He seeks to make teaching industry-relevant and to include ideas from the agile community in teaching practice. His industry experience comes from applied research with over 20 companies in his time as researcher at the Sintef research foundation. Project partners included major companies such as the energy company Equinor, the Kongsberg Group, DNV GL and consulting companies such as Sopra Steria, Bekk and Kantega.

He has further given keynote talks at the Software in Practice Conference, organised by the British Computer Society (2019), the educational symposium at the International Conference on Agile Software Development (2019) and at the International Workshop on Teamworking 21: Putting knowledge into team design (2017). He regularly gives talks to industry and at national events, and is also giving courses aimed at the software industry.

REFERENCES

[1] Devedzic, V., "Teaching agile software development: A case study," *IEEE Transactions on Education,* vol. 54, pp. 273-278, 2010.

[2] Nerur, S., Mahapatra, R., and Mangalaraj, G., "Challenges of migrating to agile methodologies," *Communications of the ACM,* vol. 48, pp. 72 - 78, 2005.

[3] Hoda, R., Salleh, N., and Grundy, J., "The Rise and Evolution of Agile Software Development," *IEEE Software,* vol. 35, pp. 58-63, 2018.

[4] Ajjawi, R., Bearman, M., and Boud, D., "Performing standards: a critical perspective on the contemporary use of standards in assessment," *Teaching in Higher Education,* pp. 1-14, 2019.

[5] Schneider, J.-G., Eklund, P. W., Lee, K., Chen, F., Cain, A., and Abdelrazek, M., "Adopting industry agile practices in large-scale capstone education," in *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: Software Engineering Education and Training*, 2020, pp. 119-129.

[6] Magana, A. J., Seah, Y. Y., and Thomas, P., "Fostering cooperative learning with Scrum in a semi-capstone systems analysis and design course," *Journal of Information Systems Education,* vol. 29, p. 4, 2019.

[7] Chatley, R. and Field, T., "Lean learning-applying lean techniques to improve software engineering education," in *2017 IEEE/ACM 39th International Conference on Software Engineering: Software Engineering Education and Training Track (ICSE-SEET)*, 2017, pp. 117-126.

[8] Nicol, D. J. and Macfarlane-Dick, D., "Formative assessment and self-regulated learning: A model and seven principles of good feedback practice," *Studies in higher education,* vol. 31, pp. 199-218, 2006.

[9] Kniberg, H., *Scrum and XP from the Trenches*, 2nd edition ed.: InfoQ, 2015.

[10] Dybå, T., Dingsøyr, T., and Moe, N. B., *Process Improvement in Practice - A Handbook for IT Companies*. Boston: Kluwer, 2004.

[11] Rising, L. and Janoff, N. S., "The Scrum software development process for small teams," *IEEE Software,* vol. 17, pp. 26-+, Jul-Aug 2000.

[12] Sutherland, J. and Schwaber, K., "The scrum guide: The definitive guide to scrum - the rules of the game," 2020.

[13] Price, M., Handley, K., Millar, J., and O'donovan, B., "Feedback: all that effort, but what is the effect?," *Assessment & Evaluation in Higher Education,* vol. 35, pp. 277-289, 2010.

[14] Ambrose, S. A., Bridges, M. W., DiPietro, M., Lovett, M. C., and Norman, M. K., *How learning works: Seven research-based principles for smart teaching*: John Wiley & Sons, 2010.