Teaching and Fostering Reflection in Software Engineering Project Courses

Håkan Burden and Jan-Philipp Steghöfer

Abstract Reflection is an important part of agile software processes as witnessed, e.g., by the Sprint Retrospectives in Scrum or by the various learning feedback loops in XP. Engineering education also emphasizes the importance of reflective practice, e.g., in Kolb's learning cycle and Schön's reflection-in/on-action. Our contribution in this chapter is a toolkit for reflective practice that shows how reflection can be used by software engineering students for two purposes: to reflect on the application of a software process and to reflect on their learning process. In order to help students understand the purpose of reflection and how to approach reflection, we follow a cognitive apprenticeship approach in which the teachers reflect about the events in the course, their own goals, and how they are aligned with the teaching. Students are asked to reflect during supervisions and as part of their written assignments from the very beginning of the course. We thus combine a meta-cognitive approach where reflection is taught as a learning strategy with a common software engineering practice of continuous improvement through reflection. We evaluate the reflective model and a course design based on it through the student, teacher, and theoretical lenses based on empirical data.

Keywords Agile · Scrum · Computer science education · Software engineering Project course · Reflective practice

H. Burden (✉)
RISE Viktoria, Gothenburg, Sweden
e-mail: hakan.burden@ri.se

H. Burden · J.-P. Steghöfer (✉)
Chalmers | University of Gothenburg, Gothenburg, Sweden
e-mail: jan-philipp.steghofer@cse.gu.se

D. Parsons and K. MacCallum (eds.), *Agile and Lean Concepts for Teaching and Learning*, https://doi.org/10.1007/978-981-13-2751-3_12

1 Introduction

Reflective practice is to evaluate your own actions and their consequences to engage in a process of continuous learning and is therefore an essential ability for professional development (Brookfield, 1995; Lyons, 2010). It enables us to not only learn from our experiences but to grow as professionals since reflection helps us challenge our assumptions and develop new professional skills as well as meta-cognitive strategies which will help us make informed decisions even when time and resources are scarce (Schön, 1983).

Despite the known benefits of reflection for professional development, there is a lack of attention within engineering education on integrating reflective practice in both courses and educational programs (Turns, Sattler, Yasuhara, Borgford-Parnell, & Atman, 2014). One of the reasons is that reflection can be intimidating since it is often perceived as sharing private thoughts and even shortcomings (Gunn, 2010). It is also challenging in the sense that reflection in some way or other asks the question of what could have been done differently. The third obstacle for teaching and learning reflective practice is that there is no clear definition of what reflection actually is.

We, therefore, explore reflective practice—from both a student and a teacher perspective—within an engineering project course and provide answers to two research questions:

RQ 1 How can we facilitate reflective practice in a software engineering project course?

RQ 2 How do students utilize opportunities for reflective practice for their continuous learning?

We will answer these research questions in the context of the development of a software engineering project course, which also shows how the outcomes directly inform a course design.

Our contribution is a toolkit for reflective practice, an artifact based on our strategy to plan, perform, and assess reflection in our course.

This chapter is structured so that first, Sect. 2 details the theoretical framework and related work and Sect. 3 describes action design research, the methodology we have chosen for our own course development. The situation as it stood when we started teaching the course is described in Sect. 4 which is then followed in Sect. 5 by a description of the toolkit for reflective action we derived from applying our methodology to the challenges we faced. Section 6 details the changes to the course after applying the toolkit. We then share both our own and the student reflections in Sect. 7 before we broaden the scope in Sect. 8 to look at how the toolkit could be applied outside software engineering. Finally, the last section concludes our reflection and points to future work.

2 Background

In this section, we will discuss the role of reflection in software engineering education and in software engineering practice as a starting point for our proposal to introduce reflection as a mainstay in software engineering project courses.

2.1 Reflection and Education

There are numerous definitions of reflection in the educational literature. Shkedi states that "*Reflection is meta-thinking (thinking about thinking) in which we consider the relationship between our thoughts and our actions in a particular context*" (2000). Smith defines reflection as "*What is in relation to what might or should be and includes feedback to reduce the gap*" (2001) while Mann defines that "*reflection is a process of inner dialogue*" (2005). Loughran summarizes the situation by stating that "*for some [reflection] simply means thinking about something, whereas for others it is a well-defined and crafted practice that carries very specific meaning and associated action*" (2002).

Schön—in describing the *reflective practitioner*—distinguishes between reflection-in-action and reflection-on-action (1983). Reflection-in-action is what we do when we encounter new situations and need to improvize on how to best proceed. Reflection-on-action is when we, later on, have an opportunity to sit down and go through the experience again in our minds to assess how it went and see what we could have done differently. On the same bearing, Freire defines *praxis* as the balance between theory and action where reflection is a means to achieve practices grounded in reflection (2000). He further states that praxis is not easy as it requires "*wise and prudent practical judgement about how to act in this situation*".

Kolb's learning cycle (2014) ties concrete experiences in a specific context to conceptualized insights from reflective practice. The transformation is facilitated by reflecting on one's own experiences to modify known concepts, generate new hypotheses and seek out what others have to say about similar situations. The gained insights are then used to set up new experiments where the hypotheses can be tested and new experiences gained.

However, applying Kolb's learning cycle will yield different results depending on who does it. Brookfield refers to this by using different lenses, where applying a new lens gives new insights (1995). For an educational setting, Brookfield defines four lenses to structure reflection for the teacher. The first lens is the autobiographical lens which is used to reflect from a personal point of view. The second lens is the student's perspective while the third lens is that of the teacher's peers. Finally, the fourth lens represents relevant theory and proven practice.

Cognitive apprenticeship (Brown, Collins, & Duguid, 1989) can be seen as an attempt to reason around the two lenses of student and teacher from the well-known concept of apprenticeship. But instead of a setting where the craft has a central role,

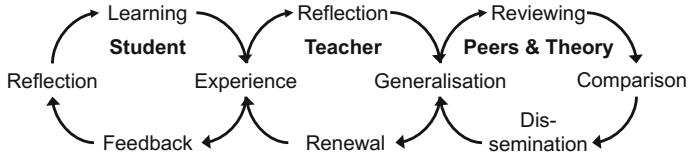

Fig. 1 The triple learning cycle, adapted from Elmgren and Henriksson (2010). Student and teacher learning is interconnected

the focus is on the cognitive skills needed for higher education. The roles of novice and master are now substituted for the roles of teacher and student as the teacher uses different techniques for the students to fathom the knowledge and skills of the master. There are six techniques:

Modeling: The master demonstrates explicitly how a task is done.
Coaching: The master supervises the novice in carrying out a task.
Scaffolding: The master sets up supportive structures to guide and help the student to experiment on their own.
Articulation: The novice uses the terminology of the trade to express new insights, their reasoning and formulate new challenges.
Reflection: The novice is given the opportunity to compare their own knowledge and skills in relation to those of the master.
Exploration: The novice is given the freedom to explore on his or her own without the interference of the master.

Being the master in such a context requires to both assess in advance how to conduct the different techniques but also to reflect-in-action to adjust them, such as in the case of scaffolding or coaching.

Inspired by Kolb, Elmgren and Henriksson (2010) represent Brookfield's four lenses in a triple learning cycle (cf. Fig. 1) where the students and the teachers meet in the concrete experience and the teacher's generalizations are shared and reviewed by peers. In this way, each cycle is informed by other cycles and knowledge is shared and spread beyond the personal cycle.

Apart from the aspects already mentioned, the literature referenced above reveals two more interesting takeaways: (1) reflection requires doing, be it in terms of experimentation (Kolb, 2014), action (Freire, 2000) or exploration (Brown et al., 1989); and (2) reflection is contextual and varies over time (Schön, 1983) and person (Brookfield, 1995).

2.2 Reflection in Software Engineering

Reflective practice is an essential aspect of software process improvement (SPI). The classic SPI loop championed, e.g., by Villalón et al. (2002), consists of the four

steps "evaluation of the current situation," "plan for improvement", "implement the improvements," and "evaluate the effect of the improvements" and thus constitutes a classic reflection loop as discussed above. Most SPI methods are built around this notion and contain approaches for some or all of these steps. For instance, inductive approaches like quality improvement paradigm (QPI)/Experience Factory (1993, 1995) and iFLAP (2008) focus on the evaluation of the current situation, the derivation of goals, and the creation of measurement plans to determine whether any changes were successful. Descriptive approaches like CMMI (2010) on the other hand focus on the planning of concrete improvement steps.

Since a structured improvement effort using one of the methods mentioned above is usually associated with a dedicated effort on the organization level and significant resources are contributed to it, long-term strategic goals are the focus of SPI (Huber, 1996). Reflection, therefore, takes place on an abstract level that encompasses a larger part of the organization and several software development efforts. The goals are often to increase long-term productivity, quality, or responsiveness to customer needs.

In contrast, iterative processes, in particular, agile ones, have a strong focus on constant change and improvement (Williams & Cockburn, 2003) and allow embedding a similar loop into each iteration and even in daily activities. An overview of such activities is provided by Babb (2014). They include group programming where reflection-in-action is practised by the people working together, estimation and planning activities, as well as the daily standups that many agile teams perform. In these cases, the reflection is implicit in the interaction of the team members and thus constitutes a social effort in communities of practice (Wenger, 1998).

Reflection is most explicitly practised in *sprint or iteration retrospectives* that are used to allow the team to discuss the current situation, identify desired future states, and devise a plan to get there. This shortens the round-trip time for improvements considerably and allows the developers to quickly try out and evaluate improvement ideas that can be helpful in the short term. Three questions are usually addressed in these short meetings: What worked well for us? What did not work well for us? What actions can we take to improve our process going forward? These questions focus the team on rather immediate issues. The reflection is therefore mostly tactical. This is in the agile spirit where immediate benefits, responding to changes, and short-term wins are emphasized over long-term strategy. However, even in such an environment, retrospectives with a broader scope that stretch all iterations for one release have been suggested as a way to reflect on the longer-term planning (Maham, 2008).

In situations in which iterations are not used (e.g., in classic waterfall projects or in projects using the V-model prominent in the automotive industry), *post-mortem reviews* are a tool that allows the development team to reflect on their work (Dingsøyr, 2005). In contrast to iteration retrospectives, postmortems have a broader scope that goes beyond a single iteration, are usually more formal and more involved, and emphasize organizational learning (Dingsøyr, 2005). Combining several post-mortems from the same organization can even help to reflect on high-level management practices that influence the effectiveness of all software development efforts in the company (Dingsøyr, Moe, Schalken, & Stålhane, 2007).

Even though the reflective practice is widespread and its positive impact is reported and empirically validated, the concrete design of the reflective activities need to be carefully tuned to achieve the desired results. The accuracy of effort estimations, e.g., is a common problem in software development projects. It was shown that it does not improve if the engineers that estimated the efforts reflected on their estimations themselves, but that improvements are only made if other professionals provide feedback (Jørgensen & Gruschke, 2009). This indicates that reflection needs to be explicitly fostered and "engineered" in order to be effective in practice.

3 Methodology

We use a modified version of *action design research* (Sein, Henfridsson, Purao, Rossi, & Lindgren, 2011), a combination of action research and design science research that focuses on designing and evaluating artifacts in a situation that "is dependent on the interaction of the participants of the research" and "can only be performed in the context of the organization and with the involvement of people within the environment under study." (Dresch, Lacerda, & Antunes, 2014, p. 94). Our modifications target the type of artifact created: instead of an ensemble of IT artifacts as described by Sein et al. (2011), we create an ensemble of teaching artifacts. This ensemble constitutes our toolkit for reflective practice in SE education (cf. Sect. 5). We describe the main characteristics and steps of action design research and how we implemented them in the following.

3.1 Action Design Research Applied to Education

Action design research is characterized by how it engages with the organization the subject of the research is embedded in. In particular, it is suitable for situations in a specific organizational setting that are addressed by intervention and evaluation within this setting. This maps very well to the educational context since we address situations that are dependent on the specific settings of the course, the students, the program, the university, etc. Solutions to challenges observed in specific courses thus constitute the artifacts that are the outcome of the methodology. They address this situation and have to be evaluated within it. The artifact is thus not only an outcome achieved by the knowledge and expertise of the researchers but heavily influenced by the users (the students and teachers in this case) and the dynamically changing situation it is designed for and evaluated in.

We follow the four stages and the associated principles suggested by Sein et al. (2011), mapping each to the educational context to which we apply the methodology. In the following, we will describe these stages and how we mapped them to our situation.

Stage 1: Problem Formulation The intended outcome of this stage is the framed problem and the theoretical premise. The two principles to follow prescribed by Sein et al. (2011) are that the research should be *practice-inspired* and that the artifact should be *theory-ingrained*. We found ourselves in a situation where there was a gap between the course's intended learning outcomes and the tools available to achieve them. This was clearly a practically relevant problem since it affected our work as teachers as well as the learning process of the students. Constructive alignment (Biggs, 1996) served as a theoretical foundation to evaluate this gap and the principle of the reflective practitioner (Schön, 1983) informed our first ideas for a solution approach. An additional, important practical component was that both involved teachers had a long-term commitment to the course and were thus not only interested in improving it in several iterations but could also expend the necessary resources to do so.

Stage 2: Building, Intervention, and Evaluation This stage's intended outcome is a realized (educational) artifact that has been evaluated and refined by use in the relevant situation. It is based on the problem formulation from the first stage. In this stage, we developed the artifact, a toolkit of reflective practices for the software engineering project to improve the course's constructive alignment. The artifact was deployed and evaluated in several iterations of the course and refined after each iteration. We followed the principle of *reciprocal shaping* since the educational artifact shapes the work with the students in the classroom which in turn has an influence on the artifact. We also ensured the *mutual learning* principle by giving the students the chance to learn from us and our attempts to use the toolkit and using the feedback from the students to learn about the effectiveness of the different tools. Finally, we adhered to the *authentic and concurrent evaluation* principle by evaluating the interventions immediately in an authentic setting.

Stage 3: Reflection and Learning The intended outcome of the reflection and learning stage is a generalized artifact that applies "to a broader class of problems" (Sein et al., 2011) achieved within a continuous learning cycle. In our context, that meant abstracting the developed toolkit from the specifics of the course and making it applicable to different course settings. We achieved this through continuous evaluation of the results in the classroom w.r.t. our goals and constructive alignment. Generality was achieved by identifying the specific issues in the course, separating them from the abstract concepts and ideas we applied, and using these concepts to identify new approaches (i.e., new tools to add to our toolkit). The artifact thus emerged guided by our reflection of the evaluation results, fulfilling the principle of *guided emergence*.

Stage 4: Formalization of Learning This stage builds upon the previous one by abstracting the generalized artifact and the problem further into design principles and the characteristics of a problem class. This stage thus allowed us to move the toolkit from the specific course instance into a broader context and abstract it to become usable for other teachers. We generalized the outcomes of the specific course to formalize the toolkit as described in Sect. 5. The abstract toolkit was then instantiated for the course iteration in spring 2017 and evaluated there to show its viability. Our final

result includes a generalization of the problem instance (software engineering project courses), a generalization of solution (toolkit), and design principles (model aspect of toolkit), thus following the *generalized outcomes* principle. The formalization of the results for dissemination is represented by this publication.

3.2 Data Collection and Analysis

In order to formulate the problem (stage 1) and to reflect and learn (stage 2) from our experience, we used a number of data sources that we analyzed repeatedly, mostly to derive qualitative data about the effectiveness of the toolkit for reflective practice and thus our teaching approach. While we evaluated data from all course iterations between autumn 2014 and spring 2017, our discussion in this chapter is focused on the latter iteration and the insights we gain from the evaluation of our current, stable version of the developed artifact.

Course evaluations We used the course evaluations conducted by the university as a tool to gauge the satisfaction of the students and identify issues with constructive alignment, workload, and cognitive demand. The evaluations consist of a voluntary, anonymous, web-based survey among the students, and a meeting with student representatives from the course itself as well as from the student union that is led by the course coordinator. We thus had the results of the survey as well as the notes from the evaluation meetings as a basis for our own analysis. We focused the analysis of the evaluation results on course development (Edström, 2008) and used it to understand which aspects the students struggled with and needed to be addressed better. As such, we attempted to receive formative feedback from the students, in particular through the discussions at the meeting.

Student reports A further source of information were the reports the students wrote throughout the course. There are three mandatory written hand-ins: the students need to reflect on how they defined their process, they need to give a report on their progress by half-time of the course, including how they refined the process, and they need to describe their overall process and lessons learned at the end of the course. In many cases, the students refer to lessons they transferred from specific teaching moments. In addition, the feedback the students got from the teachers on their reports contains connections between the reports and the individual teaching moments. We use this information to check if our toolkit is constructively aligned and yields the desired outcomes. We analyzed the reports quantitatively, applying a lightweight coding in a separate session that was independent of the grading.

Teacher notes Whenever we introduced a new teaching moment (i.e., a new tool in our reflective toolkit), we took extensive notes about the reaction of the students, whether or not we feel we achieved our objective, which questions students asked, if the time allotted was sufficient, etc. Usually, there were two sets of notes available, but occasionally only one teacher could be present during the introduction of a new tool and we relied on his notes in this case.

3.3 Threats to Validity

We discuss threats to the validity of our study and the methods used to minimize the threats, following the classification in Tomal (2010). While this classification was intended for action research, it applies well to action design research since it has a strong focus on the participants of the study, i.e., the organizational setting in which the research takes place.

One of the main threats is *differential selection*, i.e., collecting and comparing data from different groups of students in the different iterations of the course. Indeed, in different course iterations, there are changing proportions of students from different programs. In the spring iterations, most of the students come from the program on Industrial Economy, while in the autumn, most of the students are from Information Technology or Computer Engineering. The different backgrounds cause differences in how the students perceive different teaching moments and which expectations they have coming into the course. While students with a computer science background, e.g., expect a stronger technical focus and are surprised by the focus on process and reflection, industrial economy students can feel overwhelmed by the programming tasks. We addressed this threat by leveraging the longitudinal aspect of our study and trying out our tools in both settings for increased generality.

A related threat is that of *history*, i.e., differences when data is collected at different points in time. This is certainly an issue here since we combine data from different course iterations. However, we have mitigated this threat since we evaluated the data directly after the course instance in order to develop the course and our toolkit further.

Contamination, i.e., unaccounted factors outside of the study influencing its result, can be a factor here. For instance, a persistent student complaint is that the scheduling of students of different programs is not compatible, making it hard for mixed groups to find time to work together. However, due to the considerable experience of the teachers, we have a good overview of the course environment and can take such factors into account.

The threat of *instrumentation*, i.e., influences of the data collection method, can play a role, in particular since graded material was used. However, all data sources were always cross-referenced and none used in isolation. In particular, the anonymous course evaluation survey reduced the threat of bias. However, since the different data sources capture different kinds of data, a residual effect might remain.

Finally, the threat of *researcher bias* has been addressed by planning, designing, acting, and evaluating as a team. While it is possible that the team as a whole has a bias, the two teachers provide complementary viewpoints and approaches. In addition, there has been continuous exchange with program managers, the student union, students in the course, and other teachers about the course and the different attempts made to improve it.

Furthermore, there are a number of potential threats to validity that were not observed in our study: *Attrition*, i.e., the loss of participants while the study was ongoing, was not a major issue in this study since almost all students that enrolled in the course instances finished them. The *Hawthorne effect*, i.e., participants perform

better since they are given attention, is also negligible since our data collection methods are nonintrusive and only use elements that occur in the normal progress of the course anyway. While there is *maturation* of the participants during each course instance as an effect of the teaching, this is not a major concern for our study since each course instance started with a new set of students with little to no carryover from previous instances. The threat of *testing*, i.e., participants learning from pretests and thus answering differently in posttests, was also not an issue since our data sources did not include such tests. There might be a learning effect from the different student reports based on the feedback from the teachers, but this effect is intentional.

4 The Old Course Design

The starting point for our endeavor is the course instance in which the authors were first involved in the autumn of 2014. The Software Engineering Project Course represents 7.5 ECTS or 10 weeks of half-time studies and was taken by 173 students from 3 different bachelor programs run by the Computer science and Engineering department, which is a shared department of Chalmers University of Technology and the University of Gothenburg. The students formed 29 teams and collaborated with an external stakeholder in developing Android apps for truck drivers which were safe to use while driving.

The rest of this section is organized following a constructive alignment perspective where we first introduce the intended learning outcomes (ILO), learning activities and assessment tasks (Biggs, 1996) before we discuss the benefits and shortcomings of the course from a teacher and a student perspective.

4.1 Intended Learning Outcomes

The course's intended learning outcomes reflect the ambition to give an overview of what canonical software engineering is as a subject area (Burden, 2017), for instance, as defined in the Software Engineering Body of Knowledge [SWEBOK, (Bourque et al. 2014)]. Thus, the course aims regarding knowledge and understanding state that the student should be able to…

ILO1 …identify the complexities of software design and development,
ILO2 …describe the fundamentals of software engineering, such as stakeholders and requirements, and
ILO3 …describe the difference between the Customer, the Solution, and the Endeavor as well as the different methods used for each

after successfully finishing the course. In terms of skills and abilities, the student should be able to…

ILO4 …elicitate requirements from and design a solution to a real-world problem,
ILO5 …plan and execute a small software development project in a team,
ILO6 …apply skills from programming and other relevant courses, as well as
ILO7 …learn new tools and APIs on his/her own.

Finally, the students are also expected to be able to…

ILO8 …reflect on the choice of software engineering methods used throughout the project.

Following Bloom's revised taxonomy (Anderson, Krathwohl, & Bloom, 2001), the first three ILOs revolve around factual and conceptual knowledge such as basic terminology and how these relate to each other. The rest of the ILOs focus on procedural knowledge such as methods and procedures within the SE domain. The exception is ILO7 which is meta-cognitive since it requires the students to reason around their own learning.

4.2 Learning Activities

The learning activities consisted of lectures and supervision with a final presentation at the end of the course. The supervision was run on a weekly basis and students who had already taken the course were paid to supervise. The content of the supervision was supposed to target the process aspects that the student teams encountered but often revolved around tool and technology issues, such as git merging or Android debugging. There were 13 lectures which included a lecture each to introduce the course and the project scope, 4 lectures on the project-specific tools and technologies, 2 lectures on software engineering in general and 2 lectures on Scrum, 1 guest lecture, and a lecture on which tests and documentation the teams were supposed to hand in. Since the students were supposed to reflect on their choice of methods and practices but never had been given the opportunity to get feedback on their reflections, a final lecture was added while the course was running to give the students an idea of what reflection could be. During the reflection lecture, one of the authors presented his own reflections on how the course had panned out and what could be done differently for the next course instance.

The project started the same week as the course so that the second lecture introduced the project scope. This meant that the lectures regarding Scrum were given after the students had started their development effort. Subsequently, they had to make large changes to how they worked or disregard Scrum to continue working in an ad-hoc manner.

4.3 Assessment

The assessment was purely summative in terms of teacher engagement and consisted of five major elements:

Vision 30% of the grade was determined by how well the product matched the vision, the stability of the product, and the user experience;

Design Design decisions and how these were documented accounted for 10% of the final grade;

Code The code quality and the technical complexity of the solution made up 15% of the final grade;

Tests A further 15% of the final grade depended on which tests had been done and the documentation of the product;

PMR Finally, 30% of the final grade was based on a postmortem report written after the final presentation.

The five elements assessed the team performance. To be able to give the student's individual grades, the students were also asked to fill out a personal evaluation for each team member. Together with a summary of who had contributed what to the code base, this enabled the teachers to give individual students a grade that differed from the team's overall grade.

4.4 Constructive Alignment and Student Perception

The relationship between the ILOs, learning activities, and assessment tasks are visualized in Table 1. Activities and tasks can overlap: the lecture on how to use Android, e.g., relates both to ILO6 and ILO7 since it both offer an opportunity to apply skills from previous courses and learn new technologies. In the same way, the project was used to identify the complexities of software development (ILO1) and give an opportunity to execute a small software project in a team (ILO5). A shortcoming of the old course design was how the student teams needed to find ways

Table 1 The course alignment matrix for the old course design

ILO	Learning activity	Assessment tasks
ILO1	11 lectures and project	7 supervisions and terminology
ILO2	11 lectures and project	7 supervisions and terminology
ILO3	8 lectures and project	7 supervisions, vision, design and PMR
ILO4	6 lectures and project	7 supervisions, vision, design and PMR
ILO5	3 lectures and project	7 supervisions and PMR
ILO6	5 lectures and project	7 supervisions, design, code and tests
ILO7	5 lectures and project	7 supervisions, design, code and tests
ILO8	1 lecture and project	7 supervisions and PMR

to transfer the theoretical content of the lectures into practical skills during the project themselves. This was also remarked upon in the course evaluation and strategies to carry out the transfer were requested by the students. Regarding ILO7—students should be able to learn new tools and APIs on their own—the content of the five lectures centered around demonstrating how to configure the tools and make calls to the API. During the course, it became obvious that the students struggled with reflecting on their process and the decisions they took. Therefore, one of the teachers decided to add a final lecture to the schedule where he reflected himself upon the design of the course and what he would do differently if he had the opportunity to give the course again.

Regarding assessment, the only opportunity for formative feedback was during the weekly supervision slots. This relied on the student supervisors to be present and capable of handling process-related discussions. The course evaluations indicate that a recurring problem was that half of the supervisors were difficult or impossible to get in touch with and that those who carried out their supervision focused on tool related issues: "*The TA was not involved [, did not have] enough knowledge of the course or helped us in any way*". There were no assessment tasks directly related to ILO1 and ILO2. Instead, the understanding of the complexities and the fundamentals of software engineering were assessed indirectly by the terminology used by the students throughout the project and in their written deliverables.

Since the teams found it difficult to adjust their way of working to Scrum, their experiences of the Scrum practices were often superficial which led to imprecise descriptions of how they had implemented Scrum and what they learnt from their application. Instead, quite a few of the teams focused on technical descriptions of the technology they had used or the product they delivered. Furthermore, a recurring situation among those teams that did describe their process decisions focused on what happened but not alternative paths, turning the reflection into an experience report. This is also mirrored by the fact that only 3 out of 29 teams made a clear connection to the literature or the guest lectures when reflecting on their own praxis.

Among the comments given by the students in the course survey, we could see both that "*It's not easy to divide the learning goals into concrete goals which I can check if I learned*" as well as that "*It is very easy to understand what I was supposed to learn, but the course did not make it easy to learn*". Table 2 shows the student responses to selected relevant statements in the course evaluation survey.

5 A Toolkit for Reflective Practice

The artifact we developed to address the shortcomings of the course described in the previous section is a toolkit composed of different learning activities, assessment tasks, and professional practices applied by the teachers. These components are complemented by guidelines for a course structure as well as a model of reflective practice in an educational setting that provides a framework for the deployment and use of the different tools. We are going to discuss the different elements of the toolkit

in the following using the model of reflective practice as a starting point and structure for our explanations.

Our toolkit for reflective practice is not limited to teaching software processes. It is also appropriate for software architecture, testing, etc. since the ability to describe what is, what might or should be and how to bridge the gap is a useful exercise to include in all software engineering education. This is further elaborated for non-software engineering courses in Sect. 8.

5.1 Model of Reflective Practice

The model of reflective practice we developed (cf. Fig. 2) is based on reflection loops by both the student [who is the novice in terms of cognitive apprenticeship (Brown et al., 1989)] and the teacher (who is the master in terms of cognitive apprentice-

Table 2 Mean and median student responses to questions in the course evaluation on a scale from 1 to 5 where 5 is best unless otherwise noted

Statement	Mean	Median
"*I had enough prior knowledge to follow the course*"	3.63	4
"*The learning outcomes clearly describe what I was expected to learn in the course*"	3.5	4
"*The course structure is appropriate in order to reach the intended learning outcomes of the course*"	2.54	2
"*The teaching worked well*"	2.53	3
"*The assessment tested whether I had reached the intended learning outcomes of the course*"	3.22	3
"*The course administration worked well*"	2.98	3
"*The course workload as related to the number of credits was 1—too low, 5—too high*"	3.31	3
"*What is your overall impression of the course?*"	2.69	3

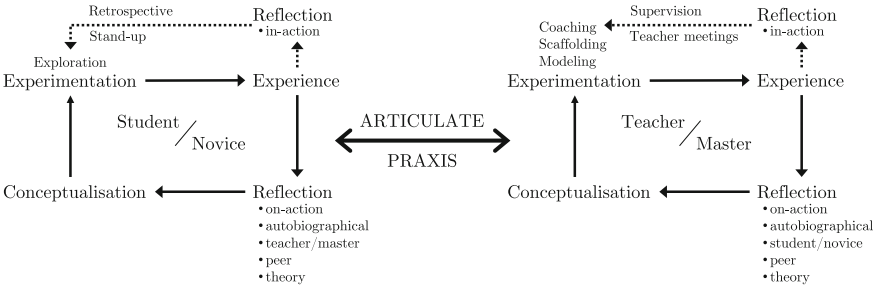

Fig. 2 A model of reflective practice, showing the different reflection loops for both teachers and students

ship). These reflection loops are connected to each other and follow the structure of Kolb's learning cycle (Kolb, 2014). Out of *experimentation* arises *experience* which is *reflected* upon either in-action or on-action (i.e., directly in the situation in which the experience is created or later on). This reflection leads to new insights that are either *conceptualised* before being used in a new round of experimentation or lead to new experimentation directly.

The *reflection-in-action loop* (Schön, 1983) shown at the top also operates on a different timescale than the lower one. For the students, the upper loop operates on the timescale of a sprint (usually about a week), while the lower loop operates on several weeks where the conceptualization is supported by deliverables in which the students document their reflection. For the teachers, the upper loop has a similar timescale to that of the students, but the lower loop operates on the scale of course instances, where conceptualization is performed after the end of each course instance and experimentation begins again with the new course. Thus, the student side represents a continuous learning process (Schön, 1983) facilitated by reflection whereas the teacher side represents in-course intervention and long-term course development.

There are numerous connections between these reflection loops. The teachers' conceptualization and its manifestation in experimentation provide the students with the opportunity to experience and enter their own reflection loop. On the other hand, the teachers observe the students' experiences and use them in their own reflection. More subtly, when teachers transition through Kolb's learning cycle from conceptualization to experimentation, they explain their own reasoning to the students and thus share their own reflections (see professional practices below) by *articulating* their *praxis* to the students. This is, in turn, a way to *model* the teachers' own reflections and serves as an example for the students in how they can reflect on their own experiences to better understand and form new concepts.

In summary, the model captures that reflection requires doing in that both students and teachers are involved in various actions and thereby also acquire shared experiences throughout the course. In addition, since reflection is conceptual it is not planned as a one-shot activity but repeatedly carried out during the course and different viewpoints are shared and considered to be able to form informed concepts (cf. Sect. 2).

5.2 Course Structure

The second part of the toolkit is the way the course is structured. In general, we try to get the students started as quickly as possible and avoid theory-laden lectures at the start. Instead, we apply the practical learning activities early on and then begin the iterative–incremental development quickly. The main part of the course is thus organized in sprints where each sprint starts with a planning session and concludes with a review and a retrospective. Students are encouraged early on to take other obligations into account when planning a sprint and set a reasonable velocity for each. Necessarily, estimations of velocity and of the user story effort are unreliable

in the beginning, but we encourage the students to learn from these mistakes and continuously improve their estimations based on their reflections. The reviews are coupled with a supervision learning activity, but are itself focused on the product and thus ideally conducted by a third party that provided the project. Students perform their retrospectives on their own, but need to record them (cf. Sect. 5.4).

5.3 Learning Activities

An important part of the toolkit is learning activities that provide students with shared experiences that they can use to develop their knowledge and skill through reflective practice. The learning activities are thus designed to trigger reflection in the students and are accompanied by specific assessment tasks that reinforce this (see below). In particular, we are utilizing three learning activities, further discussed below: a Lego Scrum simulation, a Kata for learning about scientific thinking and continuous improvement, and an exercise to teach students how to break down and estimate tasks.

To familiarize students with a modern software development process, we utilize *Lego Scrum simulations* (Steghöfer, Burden, Alahyari, & Haneberg, 2017). In these simulations, the students apply the Scrum methodology to build a Lego city based on user stories provided by the teachers. An essential element of the simulation is that the students need to reflect after each sprint and learn from their experiences. The setup of the simulations forces some issues—e.g., communication problems and a lack of planning—to come up that negatively affect the students. Through the reflective practice in the retrospectives, by reflections done by the product owner during the sprint reviews, and by reflection-on-action after the simulation, the students improve their process and approach during and after the simulation.

In order to help students understand the reflective cycle (cf. Fig. 1), we are using "*Kata to Grow*" (Rother, 2017), a simple exercise in which students apply repeated experiments to reach their goal based on the analysis of their current condition. The goal is to complete a jigsaw as quickly as possible and the students improve this process with changing constraints (e.g., "all jigsaw pieces need to be face down at the start") iteratively based on "experiments" they devise and measurements of their current state. The kata, therefore, embodies a reflection loop and shows how small improvements based on clear measurements and a defined goal can make tremendous differences. This thinking is reinforced when students are later asked to define KPIs and reflect on them in their assessment.

Finally, we are using the *Elephant Carpaccio* (Kniberg & Cockburn, 2013) exercise to demonstrate how a large assignment can be accomplished by cutting it into very thin slices—like eating an elephant. The students are asked to create small implementation tasks for a shipping cost calculator that are prioritized to deliver customer value as quickly as possible. During the exercise, the students are asked to share their reasoning in multiple iterations. After each iteration, students reformulate

the tasks. The exercise concludes with a reflection on how the exercise went, what the students learnt, and how this relates to the upcoming project.

5.4 Assessment Tasks

A particular focus of the assessment tasks is to embed each individual learning activity into the overall learning process. For that purpose, supervision sessions and hand-ins are distributed over the duration of the course. The final deliverable is used for setting the grade and should contain the content of all the previous hand-ins. This allows us to spread the workload over the course, get input for our reflection in the classroom (see below), and incentivizes the students to reflect on the learning activities right after they took place. After the Lego Scrum simulation, e.g., the students are asked to reflect on their experience and how it influences the way they set up the development process for the project.

An important ongoing assessment strategy is the *process supervision* performed by the teachers with each of the groups on a weekly basis. These supervisions are coupled with the reviews the groups conduct with the Product Owner and precede the groups' own retrospectives. In the supervisions, no technical details of the solution are discussed. Instead, the students are asked to describe their experiences with the process and associated topics such as teamwork. If students have trouble formulating issues themselves, they are nudged along those lines by questions such as "which aspect required more time than you expected?" followed by asking about the lessons learned. This triggers a reflection process that allows the teams to analyze shortcomings in the process and to have a focused discussion later on in the retrospective. It also allows the teachers to provide an outside perspective on any challenges or plans for improvement.

Since the learning outcomes state that students should demonstrate the ability to "learn new tools and APIs on his/her own", the final deliverable should contain a reflection of how well these tools worked for them. The practices the students pick up (e.g., pair programming, continuous integration, or a certain merging scheme) then become part of the reflective practice again and students evaluate their own application and their usefulness in the context of their experience. This also generates ideas on how to apply these practices better in the future, allowing students to create connections to their praxis in coming courses and their professional careers, thus lifting the learning from the current course into a larger perspective.

Furthermore, students are asked to reflect on their overall process, including the sprint reviews and retrospectives, as well as the relationship between prototype, process and stakeholder value and the relationship between their process and the theoretical literature and guest lectures. These reflections are intended to let the students reflect on the purpose of a process and how its implementation influences its effectiveness. The process should be driven by stakeholder value, result in a prototype to deliver that value, and use the different activities in the Scrum lifecycle to evaluate both the quality of the product and the quality of the process. Since the guest lectures

illuminate process issues in a context that is different from the students', it allows the students to establish whether their experience is generalizable or not and how the industrial experience differs from their own.

In the final deliverable, students are also asked to reflect on the previous deliverables and how they influenced the progress of the team and the learning. This is intended as a kind of meta-reflection to let the students reflect on their own learning process. By revisiting previous decisions and their reasoning, students are able to see the impact of their choices and how they influenced their work.

5.5 Professional Practices

Finally, the teachers themselves use reflective practice throughout the course, both in their own praxis as well as in front of the class. Reflections within the teacher group about the different course moments are in-action and allow teachers to react to emerging situations within the course. Regularly, the teachers also reflect in front of the class, thus providing an element of cognitive apprenticeships. For instance, at the beginning of a lecture, the teachers could discuss the past week or issues that have come up since they last saw the students. They would then discuss the current state, describe the state they would like to reach, and outline the plan they would like to take to get there. This kind of critical evaluation of the teachers' own work creates an atmosphere in which criticism is welcomed and students feel that their issues are being taken up. Finally, the teachers share their reflections on the course evaluation as well as on their own assessment as well as ideas and results of course development activities with the students.

6 The New Course Design

The course instance to which we applied the full toolkit took place during the spring of 2017 and was taken by 50 undergraduate students with a major in Industrial Engineering and Management and 8 students from other undergraduate programs. We use the same structure to describe the course instance as in Sect. 4 to describe the new course design.

6.1 Intended Learning Outcomes

Since the intended learning outcomes have not been identified as a prominent issue in the course, they have remained the same throughout our endeavor. Thus, the ILOs are the same as those found in Sect. 4.

6.2 Learning Activities

The learning activities are taken from our toolkit (cf. Sect. 5.3) and combined with professional practices (cf. Sect. 5.5). All activities align with our model (cf. Sect. 5.1). An overview of the course structure, how the learning activities and assessment tasks are distributed over the duration of the course, and how they relate to the model of reflective practice in Fig. 2 is given in Table 3.

The course as a whole is modeled after an iterative–incremental process that provides fast feedback to the students. A main principle to achieve this is to reduce the up-front theoretical lectures to a minimum and expose the students to practical experiences that they can use to reflect-in/on-action as quickly as possible. At the same time, these shared experiences are extremely useful in the classroom since the teachers can also reflect-on-action and help the students with the conceptualization of the experience. The other principle is to provide formative feedback to the students as often as possible in the second part of the course and to let them experience the whole reflective cycle (cf. Fig. 2) several times during the course.

The first lecture introduces the learning outcomes, the activities and the assessment tasks. We also describe what is new for this course instance based on the last course evaluation. This serves two purposes: first, it lets us communicate that we apply a reflective approach to our own course improvement; second, it lets us discuss what we came up with as concrete changes from reflecting on the course, thus employing one of our professional practices from the toolkit. As definition of reflection, we cite Smith's "*assessment of what is in relation to what might or should be and includes feedback designed to reduce the gap*" (2001). We end the first lecture by introducing Scrum in terms of roles and activities in the lifecycle.

The next scheduled activity is the Lego Scrum simulation. In the simulation, the students carry out a mini project in terms of building a Lego city (Steghöfer et al., 2017). The students go through the reflective cycle several times (once in each of four sprints) with the aim to understand and improve how they conduct Scrum. Thus we practice an agile methodology in an iterative and incremental way, where each cycle builds on the previous one and includes explicit reflective activities. The students are then asked to reflect and conceptualize their findings in D1. As teachers, we treat each sprint as one iteration of the experiment—experience—reflection-on-action cycle. In each cycle, our reflection on how we perceive the student's efforts influences our feedback to the teams during the review and the retrospective, what we want to accomplish with the next sprint, how the current exercise relates to previous exercises, and how we want to use the exercise next time around.

The practical activity of the Lego Scrum simulation is followed by a lecture where the more intricate details of Scrum as well as how to scale Scrum in a large organization is explained. The teachers relate the new theory to the experiences from the Scrum exercise and thus couple theory to practice and reflect on specific situations during the exercise together with the students.

Next up is the Kata to Grow exercise (Rother, 2017). The students are asked to complete a jigsaw as a team, iteratively improving their approach and reducing

Table 3 The course structure, outlining the different activities. Elements taken from the toolkit or using elements of the toolkit are emphasized. The relation to the elements of the model in Fig. 2 is made from the teacher perspective (*T:*) and the student perspective (*S:*). Reflection-in-action and reflection-on-action are abbreviated as RiA and RoA, respectively

Week	Type	Activity/Task	Relation to model
0	Course preparation	Preparation of material, lectures, course plan, schedule, guest lectures, etc.	*T:* RoA, Conceptualization
1	Learning activity	*Lecture: Course introduction*	*T:* Experiment.; *S:* Concept.
		Lego Scrum Simulation, *Kata to Grow*	*T:* Experiment., RiA; *S:* Whole cycle
	Assessment	Technical supervision *D1: Reflections on Lego Scrum simulation* D2: KPI	*S:*RoA
2	Learning activity	Lectures: *Scrum & Assessment*, Software Quality	*T:* Experiment., RiA; *S:* Concept., RiA
		Elephant Carpaccio	*T:* Experiment., RiA; *S:* Whole cycle
	Assessment	*Process supervision* D3: Initial product backlog	*T:* Experiment.; *S:* RoA, Concept.
3	Learning activity	Lecture: *Project background*	*S:* RoA, Concept., RiA
	Assessment	*Process supervision*	
4	Assessment	*Process supervision*	
5	Learning activity	*Guest lecture*	*S:* RoA, Concept., RiA
	Assessment	*Process supervision*	
		D4: Half-time evaluation; reflection on the work so far	*S:* RoA
6	Learning activity	Guest lecture	
	Assessment	*Process supervision*	
7	Learning activity	*Lecture: Reflections on course and project*	*T:* RoA, Experiment.;
	Assessment	*Process supervision*	*S:* Concept.
8	Assessment	Final presentations D5: Working prototype	
9	Assessment	*D6: Reflection report*	*S:* RoA
10	Course Evaluation	Feedback from students and discussions in teacher group	*T:* RoA, Concept.; *S:* Concept.

the required time. In total, the students complete the plan-act-reflect cycle six times this way. We end the exercise by sharing our reflections on the outcome and introducing the concept of key performance indicators (KPIs). The exercise concludes with a presentation of KPIs other than time that can be used for evaluating process improvement.

The third exercise, Elephant Carpaccio (Kniberg & Cockburn, 2013), shows how a large assignment can be accomplished by cutting it into very thin slices. During the exercise, the students are asked to share their reasoning at multiple intervals and receive feedback for each iteration. The exercise concludes with a reflection on how the exercise went, what the students learnt and how it relates to the upcoming project.

We follow-up with more lectures on software engineering basics, such as requirements and testing and introduce the project topic. After that, the lecture format shifts to guests from industry presenting their experiences from agile software development where each presentation is limited to the first half of the lecture. The second half is then used to reflect on how the guests' experiences resonate with those of the students and how they tie into the learning outcomes of the course. The guest lectures are in this way a possibility for the students to reflect on the praxis of a professional in relation to their own experiences.

In the last lecture, we repeat how the course evaluation has led to changes to the current course instance, how we assess the outcome and what we propose to change for the next course instance, thus highlighting our own plan-act-reflect cycle. The students are also given the opportunity to share their reflections on how the course panned out. The ILOs are then discussed with the students and they are asked to reflect on which opportunities they have had to reach the ILOs and what kind of assessment they have been given or expect to receive. The last lecture concludes by detailing the remaining deliveries and how the final presentation will be handled.

6.3 Assessment

The assessment tasks are now both formative and summative. We use the supervision slots to give the students formative feedback on their application of Scrum and discuss other aspects of the course as the teams find appropriate for their current needs. This is also inspired by agile practices: we aim to provide the students with fast and frequent feedback to adapt their behavior as they go along, instead of having to rely on a single feedback opportunity at the end of the course. The supervisions are divided into feedback on the product the students are building (modeled after sprint reviews) and feedback about the process the students apply (modeled after sprint retrospectives). The teachers are only engaged in the latter kind of feedback while the sprint review is conducted with an external Product Owner.

Assessment is also done by the teams handing in six different deliverables during the course, most of which contain elements of reflection as described in Sect. 5.4:

D1: Three reflections from the Lego exercise in terms of what the team would like to continue doing, stop doing or do differently when they apply Scrum in their project. The changes should be motivated and feasible to implement. D1 serves as the basis for a session where we select some of the reflections to illustrate how the assessment strategy will be applied. The teams also submit a social contract

detailing their ambition levels, when and how to have meetings, etc. D1 is handed in at the end of the first-course week.

D2: After the Kata exercise, the teams are asked to choose three KPIs to monitor the strategies detailed in D1. To be handed in by the end of week two.

D3: When the project scope has been introduced, the teams are asked to come up with an initial product backlog in week three. The backlog can contain epics and larger elements but should have enough user stories to fit the first sprint. D3 is then used during the Elephant Carpaccio exercises to illustrate how large stories can be split more and more thinly.

D4: Half-way through the project, in week six of the course, the teams are asked to hand in a one-page document reflecting on the work so far, both in terms of process and product. At the subsequent supervision slot, the teams pair up to facilitate the sharing of experiences and insights across teams but also to give opportunities for reflecting on each other's progress.

D5: The fifth deliverable is a working prototype for the final presentation in week eight. It does not need to be documented but it should be executable so that the students can demonstrate how they have chosen to tackle the project scope and what value they deliver to the PO.

D6: The last deliverable consists of the source code and the output from a git repository analysis tool as well as the artifacts asked for under Prototype and the Reflection Report (see below). D6 is handed in at the end of week nine which is the last week of the course.

The final team grade now relies on three elements:

Value The relevance and completeness of what is delivered in relation to how the teams have defined the scope of the project based on what the stakeholder has asked for makes up 24% of the final grade.

Prototype 30% of the final grade is based on the documentation, automatic code quality analysis, automatic and manual tests as well as design decisions.

Process Reflecting on how the team has applied Scrum as well as on the intermediate deliveries D1 to D5, describing their best practices for using new tools, and how their process relates to literature and guest lectures make up the remaining 46% of the final grade.

Just as in the old course design, we use personal evaluations and metrics from the code base to assess if there are team members that deserve a higher or a lower grade than the team grade. Deviations are never based on one source but need to be anchored in both and are often supplemented by our own observations during supervision or follow-up discussions.

6.4 Constructive Alignment and Student Perception

We supply data to show what the students report w.r.t. the course design in Table 4. Since both what we assess and how we prepare the students is different, it is not

Table 4 Mean and median student responses to questions in the course evaluation for Spring 2017 on a scale from 1 to 5 where 5 is best unless otherwise noted

Statement	Mean	Median
"*I had enough prior knowledge to follow the course*"	3.85	4
"*The learning outcomes clearly describe what I was expected to learn in the course*"	4.35	5
"*The course structure is appropriate in order to reach the intended learning outcomes of the course*"	4.25	4
"*The teaching worked well*"	4.45	5
"*The assessment tested whether I had reached the intended learning outcomes of the course*"	4.10	4
"*The course administration worked well*"	4.25	5
"*The course workload as related to the number of credits was 1—too low, 5—too high*"	3.30	3
"*What is your overall impression of the course?*"	4.20	5

possible to say what has caused the change in student perception. However, the overall increase of the scores indicates that the new course design is not seen as contradicting a good learning situation as well as supporting our view as teachers that the new design supports the students' ability to reach the intended learning outcomes.

In response to the free text question about what should be kept for the next course instance, one student replied "*The practical setup and "trial and error" approach. You learn better from making mistakes, rather than doing it right the first time.*" This implies that we still have something to work on. Not only mistakes should drive learning, but reflecting on the practical experience as a whole, including both problems that occurred and things that went well. However, mistakes tend to force reflection since the identification of the mistake resonates with the description of what is. If the mistake is to be corrected, a change is needed which also encourages to consider what should be and feedback to reduce the gap. But success is also an experience worth reflecting over since understanding what enabled the success and how it can be repeated saves both effort and time in the future.

In terms of constructive alignment, comparing Table 1 with Table 5 shows that the intended learning outcomes are now addressed with additional exercises, thus emphasizing skill development and practical experience instead of a mostly theoretical approach. Since these exercises are always connected to reflections, this element is significantly strengthened accordingly.

In relation to ILO7—students should be able to learn new tools by themselves—the lectures and exercises do not mention how to use the new tools but reflect on what the students have experienced during the exercises and how that can be transferred to the project. The exercises also present teachers and students with shared experiences that can serve as basis for reflecting together as illustrative examples to explain concepts and strategies for handling these. For instance, the students played with

Table 5 The course alignment matrix for the new course design

ILO	Learning activity	Assessment tasks
ILO1	6 lectures, 2 exercises and project	5 supervisions, D2 and terminology
ILO2	6 lectures, 2 exercises and project	5 supervisions and terminology
ILO3	6 lectures, 2 exercises and project	5 supervisions, process and terminology
ILO4	4 lectures, 2 exercises and project	5 supervisions, D3 and prototype
ILO5	3 lectures, 3 exercises and project	5 supervisions, D1, D2, prototype and process
ILO6	3 lectures, 3 exercises and project	5 supervisions
ILO7	1 lecture, 3 exercises and project	5 supervisions and process
ILO8	4 lectures, 3 exercises and project	5 supervisions, D1, D4 and process

Lego as kids but still struggle with finding the right Lego pieces for their buildings. This shared experience is something we can go back to as we reflect on how their programming skills might transfer to using a new API and development tools.

7 Reflections on the Toolkit

This section will first detail how the students utilized the opportunities for reflection-in- and -on-action, before we describe our own thoughts and relate those to existing literature.

7.1 Student Lens

Immediate reflection-in-action is relevant as events unravel during the course and students need to handle situations for which they are not prepared. These reflections are sometimes difficult to document due to the time and place when and where they occur. However, the students have recurring opportunities to reflect on their experiences, e.g., during daily stand-up meetings or sprint retrospectives. These opportunities are easier to document and reflect on since they occur at defined points during the sprints and allow to define what should be and how to bridge the gap while the project still runs. The teams' reflection-on-action is documented in the reflection report after the final presentation (D6), meaning that the students do not have the possibility to implement their suggested feedback within the course. Instead, the intention is that the insights will be of use in their future studies and professional life.

Reflection-in-Action Two teams decided to structure their reports to mirror Smith's definition of reflection by first describing what they did, and then described what they would do differently and how. The first team consistently used the subheadings "*The*

situation as it is" and "*What we would like it to be*" where the latter also included strategies for realizing the change. The other team defined what went well, what could improve and how they could improve for each of the bullets required for the reflection report. As an example, they stated that their communication with the PO went well, that they could improve in how they used roles within the team and that the improvement could be realized by not only assigning responsibilities but also defining what each responsibility covered.

One of the student teams wrote in their reflection report that they would include how to conduct daily Scrum meetings in their social contract. Since the team members took different parallel courses, they had difficulties finding a time that suited everyone. They therefore suggested to regulate how all team members can participate even if they cannot be physically present upfront. Another team stated that it was difficult in the beginning to keep the meetings short and concise since when a team member described what they had done, other members wanted to know how a specific task had been solved. The meetings, therefore, tended to involve lengthy technical descriptions. The team came up with two strategies to shorten the meeting time. First, they decided to stand up during the meetings since this improved focus and was recommended by literature. Second, they planned meetings where insights regarding how to handle new tools and technologies could be shared. Their conclusion was that while it is important to share information it is also important to know when to share what.

A similar experience was reported by a third team in relation to the sprint retrospectives. In the beginning of the project, these were held at the supervision slot and as a consequence just after the retrospective and before the planning as well as in the same location. The discussions quickly became technical and focus shifted from process to product. Therefore, they decided to have the retrospectives at another physical location and ban visible computers. In this way, the focus on process improved and the team reports that their satisfaction with the retrospectives increased over time.

Reflection-on-Action Regarding the peer lens, one team stated that it was helpful to see how another team handled the same challenges they faced. However, they did not provide details about the challenges and what they could have done differently. Other teams were more articulate but concluded that the peer discussion came a week too late for them to have a real impact. By the time they were asked to reflect on their first two deliverables, they had just overcome a major obstacle in how to communicate with the shared backend. Therefore, they felt that the rest of the sprints would be more straightforward and would allow the team to focus on delivering value instead of debugging. This gave them the opportunity to assess what lay ahead and to evaluate what they just had done in relation to what they thought they would do.

In relation to the first delivery (D1), one team felt that they were initially right in stating the importance of understanding the needs of the product owner (PO) instead of the desired solution since there might be other ways of delivering value: "*Focus was on how our sketch and vision could be adapted to the PO's instead of understanding why the PO came with a specific solution*". Half-way through the project the team managed to shift focus and concentrate on the context of the PO and

from there redirect their development effort towards a system more suitable for the needs of the PO.

An example of how a team identified their own learning progress throughout the project relates to the definition of done that they used for their user stories: "*As our understanding of the system and project grew, it became easier to identify and structure these definitions.*" As we saw in relation to daily Scrums and the introduction of meetings with the specific purpose of sharing knowledge between team members, the peer lens was also applied within teams to share knowledge and reflect on how to improve their way of working.

An interesting observation is that while all teams relate their reflections on the course literature and the guest lectures, none of the teams relate their reflections to what the teachers have said.

7.2 Teacher Lens

The toolkit for reflective practice proposed in Sect. 5 allowed us to address the gap between what we imagined the course to be and what it was. It is the result of a 3-year effort to improve the course and move it from a product-focused programming project with poor constructive alignment and a mismatch between theory from lectures and what was practically applied towards a process-focused engineering course that is driven by practical experience and continuous reflection.

Reflection-on-Action Our own perception of the course has improved significantly with the introduction of the different elements of the toolkit for reflective practice. While the course is still known amongst the student body as "the android course", we are confident that we now focus on the process issues that is at the heart of the intended learning outcomes much better. This also makes it easier for us to communicate our vision for the course to the students. The expectations of the students and the place of the course in the different programs are also much clearer. Instead of being yet another development project, the course now offers different and novel content.

In relation to the old course design, the project now starts on the third week of the course. Instead of letting the students immediately get to work on the project we use the first weeks to introduce the central concepts and Scrum. These concepts are then explored during three exercises where each exercise has a component of reflection and feedback. This change means that there are fewer supervisions but also that the students get help in bridging the gap between theory presented in the lectures and the practice they are asked to explore during the project as well as an opportunity to reflect on what they have done during the exercise and what they want to do during the project.

Including collective feedback into the lectures also means that we as teachers not only have the opportunity to motivate the exercise and the deliverable, we can also reflect on what went well with the exercise and how we aim to improve it for the next time. We thereby verbalize our own reflection in front of the students and

model how we came to give the exercise the way we did. In this way, the exercises supply a scaffold for the students to reflect on how they plan, execute, and evaluate a team project as well as agile practices like splitting user stories into tasks. Since supervision is handled by the teachers with a deliberate focus on process matters they provide opportunities to coach the students in their reflective practice based on the ideas, uncertainties, and milestones they want to bring up. We as teachers can also bring up topics we find worth discussing. Throughout the different activities, we can go back and relate what is happening and how we reflect within the current context to the shared experiences we obtained through the three exercises. In this way, our new course design mirrors the recommendations to combine subject matter with reflective practice so that the task becomes more concrete and has an immediate bearing on the students' professional development (Mathiassen & Purao, 2002).

An important aspect of this new structure and the progression of assessment tasks is that we are able to build a trustful relationship with the students (Gunn, 2010). Since they have the opportunity to receive formative feedback continuously but only the final hand-in is graded, they understand our expectations and how they can address them much better. Trust is also built by articulating our own reflections and being open about problems in the course and how we are going to address them. We thus demonstrate that failure is an opportunity to learn and that admitting mistakes is an important step in the learning process. We thus allow a cognitive apprenticeship to form in the classroom.

Reflection-in-Action Having reflected on what we do and how we want to improve the course gives us an understanding of what we want to achieve with the different learning activities and the corresponding assessment. At the same time, we also gain new experiences each time we give the course. By sharing these experiences and how we acted and reasoned provides us teachers with a portfolio of situated reflections that we can rely upon when we encounter situations for which we are not prepared. In this way, reflection-on-action supports our reflection-in-action.

7.3 Theoretical Lens

When comparing our own work with related literature, it becomes evident that reflective practice is a recurrent theme in software engineering education. In the work of Hazzan, e.g., reflection is seen as a driving factor in education about human factors in software engineering (2004). The same author also suggests to use reflection with a tutor as a way to continuously drive a project forward in a studio environment (Hazzan, 2002). However, Hazzan couples reflective practice in software engineering education directly to the specific method of the studio in which students meet with a capable tutor several times a week, thus increasing commitment and motivation and exposing the students to constructive criticism and different social interactions connected to collaborative work. While this method is very intriguing, it also requires significant resources, both in terms of meeting space and effort by the tutors. Such a

method is thus not feasible in the resource-constrained environment we find ourselves in.

Another take on the studio as an instrument for reflection is presented by Bull and Whittle (2014). They argue that project-based courses are better for facilitating reflection than lecture-based courses since they give students the opportunity to work iteratively. Still, such courses often suffer from considering reflection as an implicit learning objective and do not explicitly address it through the teaching activities. The authors conclude that the studio approach is recommendable for fostering reflection at program level and allows addressing learning objectives over multiple courses. We agree, while we also believe that our own course is an example of how a single course can introduce explicit learning objectives, activities, and assessment strategies that foster student reflection.

The studio method championed by Hazzan and Bull and Whittle is one example of the more abstract concept of *communities of practice* (Wenger, 1998). They regard learning as a social and collaborative effort that is based on the common passion for a subject and the interaction between the learners and the teachers. Our toolkit for reflective practice helps us in establishing such a community of practice: joint activities in the course within a common domain create a community that is based on practical experience and reflective practice about this experience. Continuous interaction between the students and between students and teachers and the learning activities are designed to create a "shared repertoire of resources" that helps the students in their learning process and in achieving the intended learning outcomes.

Reflection has also been acknowledged as a problem-solving strategy in software engineering education. For instance, teaching students how to reflect in order to improve their skills in writing software tests (Edwards, 2004) enables them to move away from a trial-and-error approach and thus allows them to find solutions more quickly and efficiently. In particular, the role of feedback for the success of reflective practice is emphasized in Edwards (2004). While this feedback is provided by an automated system in the course the contribution reports on, we aim to provide formative feedback in our supervision sessions with the students and in the different learning activities throughout the course.

In terms of assessment, the assessment strategies in our toolkit for reflective practice are instantiated, among others, in the final deliverable that contains a reflection report in which students reflect on their experience with the process. This is similar to the use of postmortem reports to evaluate software architecture projects suggested by Wang and Stalhane (2005). Such reports are often used in the industry to analyze a product development effort and draw conclusions that can support a software process improvement initiative (Dingsøyr, 2005). However, the cited paper proposes to only include positive and negative experiences in the report. While this is an important part of reflective practice, the crucial part of deriving concrete improvement steps and evaluating those in practice—an essential part of our toolkit for reflective practice—is missing.

8 Applying the Toolkit Outside SE Education

While we developed the toolkit for use in a software engineering project course, its general outline should be applicable to different course structures within engineering and the sciences in general. The model of reflective practice (cf. Sect. 5.1) is independent of concrete course content and only mandates an iterative approach. The assessment tasks (cf. Sect. 5.4) we use are likewise independent of the concrete product or discipline of engineering and focus on reflecting on the students' praxis and choices. Similarly, the professional practice of the teachers (cf. Sect. 5.5) of reflecting about the course amongst themselves and in front of the class is completely independent of the concrete discipline being taught.

We see the main application area in engineering project courses in which an artifact needs to be developed by students following a specific process. In these situations, students often exhibit a "product over process" attitude (Steghöfer et al., 2016). Using reflective practice that is focused on the process draws the students' attention to these issues and makes it easier for the teachers to put process aspects into the foreground. The course structure (cf. Sect. 5.2) is applicable in such project courses with minimal modifications based on the background knowledge and skills students need to acquire before being able to start working on the product and the length of the course. The learning activities (cf. Sect. 5.3) might also be adapted. While the kata and the process supervision are transferrable to other disciplines, the Lego Scrum exercise uses a dedicated *software* development process. However, other simulations or serious games could be used to achieve a similar effect. One example is the urban planning game described in Mayer, Carton, de Jong, Leijten, and Dammers (2004) in which students simulate the development of a city and the necessary negotiations between the involved stakeholders.

9 Conclusions

In this chapter, we have described the toolkit for reflective practice, a set of teaching, assessment, and professional practices based on a model of reflective practice for engineering courses with a particular focus on software engineering. We have shown how the toolkit was developed using action design research based on issues observed in a project course we teach and how the toolkit is applied in the current version of the course. The toolkit is thus an answer to RQ1: *How can we facilitate reflective practice in a software engineering project course?* Our discussion of the student perception, our own perception and the relation to other published work also shows that the toolkit is viable, thus providing an answer to RQ2: *How do students utilize opportunities for reflective practice for their continuous learning?*

It is important to note that our toolkit for reflective practice contains many aspects that allow us to teach how reflection actually works. Being able to reflect is a skill that needs to be acquired by our students. Our experience shows that students are

successful in doing this by following the teacher's example and by being encouraged to reflect continuously while the course is running.

In the future, we would like to make reflection of both students and teachers an even more prominent feature of the course. One way to achieve this is to instate daily stand-up meetings, a practice that many student groups already take up on their own. Another would be to start each lecture could with a reflection by the teachers. At the moment, this only happens if there are events that make it prudent to do so. A further possibility is to include additional opportunities for peer assessment in the course, where students perform peer reviews of the reflection reports of the other students to see positive and negative examples. Notably, being able to write a good review is another skill that we cannot expect from our students. Thus, reviewing would have to be introduced and formative feedback on the reviews would be necessary. However, since architecture and code reviews are common practices in software engineering, this could provide an additional opportunity to include an important professional practice in the course.

References

Anderson, L., Krathwohl, D., & Bloom, B. (2001*). A taxonomy for learning, teaching, and assessing: A revision of Bloom's taxonomy of educational objectives*. Longman.

Babb, J., Hoda, R., & Nørbjerg, J. (2014, July). Embedding reflection and learning into agile software development. *IEEE Software*, *31*(4), 51–57. https://doi.org/10.1109/MS.2014.54.

Basili, V. R. (1993). The experience factory and its relationship to other improvement paradigms. In *European Software Engineering Conference* (pp. 68–83). Springer.

Basili, V. R., & Caldiera, G. (1995). Improve software quality by reusing knowledge and experience. *MIT Sloan Management Review*, *37*(1), 55.

Biggs, J. (1996). Enhancing teaching through constructive alignment. *Higher Education*, *32*(3), 347–364.

Bourque, P., Fairley, R. E., et al. (2014). *Guide to the software engineering body of knowledge (SWEBOK (R)): Version 3.0.* IEEE Computer Society Press.

Brookfield, S. (1995). *Becoming a critically reflective teacher*. San Francisco: Jossey-Bass.

Brown, J. S., Collins, A., & Duguid, P. (1989). Situated cognition and the culture of learning. *Educational Researcher*, *18*(1), 32–42.

Bull, C., & Whittle, J. (2014, July). Supporting reflective practice in software engineering education through a studio-based approach. *IEEE Software*, *31*(4), 44–50.

Burden, H. (2017). DAT255 Software Engineering Project, HT2014. Retrieved March 29, 2018, from https://github.com/hburden/DAT255/tree/ht2014.

CMMI Product Team. (2010). CMMI for development, version 1.3 (tech. rep. No. CMU/SEI-2010-TR-033). Software Engineering Institute, Carnegie Mellon University.

Dingsøyr, T. (2005). Postmortem reviews: Purpose and approaches in software engineering. *Information and Software Technology*, *47*(5), 293–303.

Dingsøyr, T., Moe, N., Schalken, J., & Stålhane, T. (2007). Organizational learning through project postmortem reviews—An explorative case study. *Software Process Improvement*, 136–147.

Dresch, A., Lacerda, D. P., & Antunes, J. A. V. (2014). *Design science research: A method for science and technology advancement*. Springer Publishing Company, Incorporated.

Edström, K. (2008). Doing course evaluation as if learning matters most. *Higher Education Research & Development, 27*(2), 95–106. https://doi.org/10.1080/07294360701805234. eprint: http://dx.doi.org/10.1080/07294360701805234.

Edwards, S. H. (2004). Using software testing to move students from trial-and error to reflection-in-action. In *Proceedings of the 35th SIGCSE Technical Symposium on Computer Science Education, SIGCSE '04* (pp. 26–30). Norfolk, Virginia, USA: ACM. https://doi.org/10.1145/971300.971312.

Elmgren, M., & Henriksson, A. (2010). *Universitetspedagogik*. Norstedts.

Freire, P. (2000). *Pedagogy of the oppressed: 30th anniversary edition*. Bloomsbury Academic.

Gunn, C. L. (2010). Exploring MATESOL student 'resistance' to reflection. *Language Teaching Research*, *14*(2), 208–223.

Hazzan, O. (2002). The reflective practitioner perspective in software engineering education. *Journal of Systems and Software*, *63*(3), 161–171.

Hazzan, O., & Tomayko, J. E. (2004, March). Reflection processes in the teaching and learning of human aspects of software engineering. In *17th Conference on Software Engineering Education And Training, 2004. Proceedings* (pp. 32–38). https://doi.org/10.1109/CSEE.2004.1276507.

Huber, G. P. (1996). Organizational learning: A guide for executives in technology-critical organizations. *International Journal of Technology Management*, *11*(7–8), 821–832.

Jørgensen, M., & Gruschke, T. M. (2009). May). The impact of lessons-learned sessions on effort estimation and uncertainty assessments. *IEEE Transactions on Software Engineering*, *35*(3), 368–383. https://doi.org/10.1109/TSE.2009.2.

Kniberg, H., & Cockburn, A. (2013). Elephant Carpaccio exercise. Retrieved October 30, 2017, from https://docs.google.com/document/d/1TCuuu8Mm14oxsOnlk8DqfZAA1cvtYu9WGv67YjsSk/pub.

Kolb, D. A. (2014). *Experiential learning: Experience as the source of learning and development* (2nd ed.). FT Press.

Loughran, J. J. (2002). Effective reflective practice in search of meaning in learning about teaching. *Journal of Teacher Education*, *53*(1), 33–43.

Lyons, N. (Ed.). (2010). *Handbook of reflection and reflective inquiry—Mapping a way of knowing for professional reflective inquiry*. New York, NY: Springer.

Maham, M. (2008, August). Planning and facilitating release retrospectives. In *Agile 2008 Conference* (pp. 176–180). https://doi.org/10.1109/Agile.2008.60.

Mann, S. (2005, July). The language teacher's development. *Language Teaching*, *38*, 103–118.

Mathiassen, L., & Purao, S. (2002). Educating reflective systems developers. *Information Systems Journal*, *12*(2), 81–102.

Mayer, I. S., Carton, L., de Jong, M., Leijten, M., & Dammers, E. (2004). Gaming the future of an urban network. *Futures*, *36*(3), 311–333.

Pettersson, F., Ivarsson, M., Gorschek, T., & öhman, P. (2008, June). A practitioner's guide to light weight software process assessment and improvement planning. *Journal of Systems and Software, 81*(6), 972–995. https://doi.org/10.1016/j.jss.2007.08.032.

Rother, M. (2017). Kata to grow–A simple, free exercise to help teach scientific thinking [online]. Retrieved October 30, 2017, from https://www.katatogrow.com/.

Schön, D. A. (1983). *The reflective practitioner: How professionals think in action*. Harper torchbooks. Basic Books.

Sein, M. K., Henfridsson, O., Purao, S., Rossi, M., & Lindgren, R. (2011). Action design research. *MIS Quarterly, 35*(1), 37–56. Retrieved from http://www.jstor.org/stable/23043488.

Shkedi, A. (2000). Educating reflective teachers for teaching culturally valued subjects: Evaluation of a teacher-training project. *Evaluation & Research in Education*, *14*(2), 94–110.

Smith, R. A. (2001). Formative evaluation and the scholarship of teaching and learning. *New Directions for Teaching and Learning*, *2001*(88), 51–62.

Steghöfer, J.-P., Burden, H., Alahyari, H., & Haneberg, D. (2017). No silver brick: Opportunities and limitations of teaching Scrum with Lego workshops. *Journal of Systems and Software*, *131*, 230–247.

Steghöfer, J.-P., Knauss, E., Alégroth, E., Hammouda, I., Burden, H., & Ericsson, M. (2016, May). Teaching agile—Addressing the Conflict between project delivery and application of agile. In

Software Engineering Education and Training Track, the 38th International Conference on Software Engineering, Austin, TX.

Tomal, D. R. (2010). *Action research for educators.* Rowman & Littlefield Publishers.

Turns, J., Sattler, B., Yasuhara, K., Borgford-Parnell, J., & Atman, C. J. (2014). Integrating reflection into engineering education. In *Proceedings of the ASEE Annual Conference and Exposition*. ACM.

Villalón, J. A. C.-M., Agustín, G. C., Gilabert, T. S. F., Seco, A. D. A., Sánchez, L. G., & Cota, M. P. (2002). Experiences in the application of software process improvement in SMES. *Software Quality Journal*, *10*(3), 261–273.

Wang, A. I., & Stalhane, T. (2005, April). Using post mortem analysis to evaluate software architecture student projects. In *18th Conference on Software Engineering Education Training (CSEET'05)* (pp. 43–50). https://doi.org/10.1109/CSEET.2005.42.

Wenger, E. (1998). *Communities of practice: Learning, meaning, and identity*. Cambridge University Press.

Williams, L., & Cockburn, A. (2003, June). Agile software development: It's about feedback and change. *Computer, 36*(6), 39–43. https://doi.org/10.1109/MC.2003.1204373.