Supporting Reflective Practice in Software Engineering Education through a Studio-Based Approach

Christopher N. Bull and Jon Whittle, Lancaster University

// Learning is a lifelong process, especially in the fast-paced software industry. In addition to formal training courses, good software developers continually learn by reflecting on what they’ve done in the past. However, reflective practice is rarely taught explicitly in university software engineering education. One way to teach reflective techniques from the start is through studio-based learning. //
without actually doing it. This gives credence to why it’s important to practice software engineering.

So why, then, if reflective practice is so valuable, do so many forms of software education not always support it? Lecture-based delivery, for example, can suppress reflection due to its didactic style. Project-based courses are better in this regard because they give students the opportunity to rework solutions over time. However, such courses don’t guarantee that they will “build in” reflection, primarily because reflection is often only considered implicitly in learning goals.

In this article, we argue that reflective techniques should be more explicitly considered in software engineering education. In fact, all forms of software engineering education can encourage reflection. Lecture-based teaching, for example, albeit a mode that’s “primarily one way,” can still offer opportunities for reflection, especially if the educator makes the lecture interactive or if the lecture is accompanied by practical lab sessions. Project-based courses further encourage reflection: group-based projects, in particular, offer many opportunities for reflection.

In practice, different pedagogical styles vary in their support for reflection—any method can include reflective practice, but some support it more naturally than others. We argue that a particularly appropriate method for instilling reflective practice is through studio-based education. Studio education originates from disciplines such as architecture, design, and art. A studio is a room for supporting collaborative and individual learning that’s very practical and often project-based, employs coaching (as well as formal and informal critiques), and is commonly said to act as the student’s second home: “Students do not so much attend these events as live in them.” Conceptually, a studio “is a process of learning by doing,” where students can play roles with their peers in addition to what the staff would also normally perform, such as coaching and critiquing. A key difference to traditional teaching is that lectures are rarely used, if at all.

There are numerous understandings and implementations of studio education, but through interviews with experienced studio educators in architecture, design, and art, we’ve recognized several key elements, such as how the physical environment and culture affect a studio.

**STUDIO EDUCATION**

Studio education originates from disciplines such as architecture, design, and art. A studio is a room for supporting collaborative and individual learning that’s very practical and often project-based, employs coaching (as well as formal and informal critiques), and is commonly said to act as the student’s second home: “Students do not so much attend these events as live in them.” Conceptually, a studio “is a process of learning by doing,” where students can play roles with their peers in addition to what the staff would also normally perform, such as coaching and critiquing. A key difference to traditional teaching is that lectures are rarely used, if at all.

There are numerous understandings and implementations of studio education, but through interviews with experienced studio educators in architecture, design, and art, we’ve recognized several key elements, such as how the physical environment and culture affect a studio.

**References**


**Reflection in Studios**

The aspect of studio education that we’ve found to be most important to its success is the culture that the students and staff foster. Elements of this culture include a sharing ethos, being social, treating the space like a second home, maintaining a good work ethic, utilizing peer learning, and accommodating serendipity into interactions and learning.

Studio culture is also important to reflective practice because it provides many opportunities that allow for a valuable element of being reflective: constant questioning. The notion of constant questioning, first described in the context of software studios by James Tomayko at Carnegie Mellon, is a “method of personal reflection and external criticism.” It’s a broad idea that can be encompassed within many activities to support reflection, and can occur both implicitly and explicitly. Assumptions should be questioned and ideas or solutions articulated to a fresh pair of ears. What is important here is that constant questioning is integral to the entire process of the course, not just a report handed in at the end of a piece of work—something that
we believe involves less reflection and has a lower impact on students because the work is already complete.

Looking at publications about software studios from a number of universities, we’ve identified some examples of reflective activities:

- constant questioning,
- open-ended and complex problems,
- rapid iterations of design solutions,
- culture of critique,
- teamwork,
- collaborative learning,
- interactive problem solving,
- student presentations,
- peer review, and
- coaching.

This isn’t an exhaustive list, but we believe that these activities offer opportunities to reflect. Furthermore, they don’t need to occur within a studio context, but studio-based education is an example of where they occur more often than not, due to the emphasis on the small-scale, face-to-face communication and relationships common within a studio. Traditional teaching could accommodate some of these, but in reality, the didactic nature of lecturing would likely not encourage their use.

Project-Based Learning vs. Studio

We’re frequently asked whether project-based learning (PBL) in software engineering education already provides the benefits of studios. This is actually a relatively easy question to answer, but depends on how you see a project-based course; confusion can come from the fact that definitions of PBL can be quite broad and varied. We define PBL as real-world problems explored in a problem-solving context. A summary of various definitions is also provided by John Thomas: “projects are complex tasks, based on challenging questions or problems, that involve students in design, problem-solving, decision making, or investigative activities; give students the opportunity to work relatively autonomously over extended periods of time; and culminate in realistic products or presentations.”

PBL shares several similarities and benefits with studio education, although when properly administered, a studio can significantly improve learning over PBL. It can allow for improved reflection over traditional teaching, but we feel that there’s an even greater opportunity for reflective practice in studios. Some examples of this can be observed through the use of the culture of critique and collocation with other students; these can occur in a PBL course, but they aren’t required to. Another example is that PBL doesn’t necessarily require a physical space that the students can call home. For many project-based courses, students meet infrequently, at pre-arranged times, which is in contrast to a studio where student interactions are constant, spontaneous, and serendipitous. A dedicated physical space also affords the opportunity for students to produce design work that remains up on whiteboards or studio walls. Students can constantly refer back to their work, which isn’t possible in a classical PBL course because students don’t have their own space.

Observation Methods

We observed our studio implementation for a variety of reasons, including reacting to the course as it progressed and enacting changes if necessary, but primarily to see how our studio stacked against other implementations. The main techniques we used to explore the studio implementation at Lancaster University during the 2012–2013 academic year were

- observation (one of us formally observed students in the studio), and
- focus group (we conducted semistructured discussions with students about their experiences in the studio; some questions originated from observations made throughout the year).

Our research was also reinforced by interviewing instructors from other disciplines—looking at what a studio really is in its native context (architecture, design, and art).

We started our software studio at Lancaster University in 2012 for the second and third years of the software engineering degree. The degree shares “core” modules with computer science degrees, but all unique software engineering modules are housed in the software engineering studio. Students have 24-hour
access to the studio and are encouraged to use the room outside of any timetabled sessions, known as workshops, and to treat it like they own the space. The studio modules house year-long, student-defined group projects that involve a 3.5-hour workshop session to be held in the room every two weeks. These workshops are practical in nature, with students performing set activities as part of a particular stage of their project (for example, architecture design or testing strategies), incorporating the practical activities directly into their projects (such as practicing peer-testing concepts, and then performing them on each other’s project), and often presenting the output of the task or what they’ve learned to other staff and students in the studio.

As students define their own projects with staff guidance (ensuring suitable coverage of curriculum), the project’s concept and goals provided them with a greater sense of ownership of the course. It also means that all groups work on vastly different projects, which therefore aren’t sandboxed; students have the opportunity to take their projects further and deploy their solutions outside of the course if they desire, which was a driving factor for several of the students. The projects are monitored by staff through weekly supervision meetings, as well as by all staff present during each workshop session, to ensure that the curriculum is covered in each project.

The studio has a high staff-to-student ratio, with a typical workshop containing anywhere between one to four lecturers, two teaching assistants, and 10 students per year. Any large-scale expansion would require further research to determine the effect on staff resources. Although the studio appears to require a high level of staff input, it’s balanced by the reduced time required of lecturers, both in preparation and attendance. Studios also encourage lecturers to focus on flexible in-session action rather than preparation of materials beforehand.

**What a Reflective Studio Looks Like**

Our observations provided interesting perspectives on reflective practices. Sometimes reflection was planned or expected as part of an activity, but frequently it was due to the nature of the studio and its activities. Of course, not all instances of reflection need to occur within a studio, but the nature of studios and the culture they encourage naturally allow reflection to occur. Several of the examples we observed illustrate a variety of reflection-in-action and reflection-on-action practices.

**Mentors.** Each group had an assigned supervisor who coached the group. The group met with the mentor at least once a week to articulate the project’s problems and solutions—the purpose was to provide guidance to the project and students, and to keep them on track with any course requirements. Mentors also constantly questioned the students, encouraging them to reflect. So, for example, rather than suggesting an alternative, mentors would pose questions to students, encouraging them to generate their own possible or alternative solutions.

**High staff-to-student ratio.** We believe that a higher staff-to-student ratio supports better reflection during studio workshop sessions because of the various perspectives each staff member brings to the studio—take a project proposal to an architect, and she’ll discuss the architecture, take it to a cybersecurity researcher, and he’ll discuss security and the robustness. Having more staff available during workshop sessions led to plenty of opportunities for rich feedback and discussion, feeding back into the reflective process of constant questioning.

**Large and complex projects.** Identifying solutions to large and complex problems, those that you’re expected to experience as a practitioner, isn’t supposed to be trivial. Project scale and complexity will inherently encourage reflection because it’s difficult to determine what the “right” solution is: Does the solution complete the project’s goals? How well does it complete them? Reflective conversations were observed within groups as a direct result of the project’s size, for example, discussing how to better manage the development of a subsequent milestone compared to a previous one. Large projects can provide an increased opportunity to experience big problems and creative problem solving.

**External dependencies.** Our year-long projects encourage students to
learn more than just the scope of the curriculum. Because students define the projects, they’re forced to further reflect on their practice. One example from our studio was a group of students who needed to set up a server for a website. They found issues with requesting server access and setting up the virtual machines; subsequently, they missed some deliverables. This taught them to identify and organize resources early, as well as to have mitigation plans in place. It was a valuable learning experience, and they did well at the end of the year.

**Presenting.** Throughout the studio course, the students gave several presentations and demonstrations of their work and solutions to the staff and other students within the studio. These short, often semiformal, presentations provided further opportunities for them to engage with each other, encouraging critique across groups and providing yet another avenue for constant questioning. Most questions weren’t aimed at simply how students performed tasks but were crafted to ask “how” and “why.” The audience was also encouraged to reflect on the presentation and questions, which we observed when students incorporated that interaction into their thinking, such as seeing how another group approached a particular problem.

**Visible work.** Because the studio was our dedicated space, students could leave their work up, with designs and prototypes remaining on walls or whiteboards. The benefit to other people in the studio was in seeing other groups’ progress and how their thought processes emerged around a particular problem. Making work visible also allows for greater possibilities of reflection-on-action for students’ past work and broader reflection-in-action for their current activity, primarily because it increases the accessibility to their work and encourages feedback from everyone else in the room. Because each group works on vastly different projects, there’s no culture of hiding your work for fear of plagiarism, inadvertent or otherwise. In fact, looking at each other’s work is encouraged.

**Whiteboards.** We had enough mobile whiteboards for at least one board per group, which removed any pressure to wipe content between sessions. Whiteboards frequently led to group discussion, which is conducive to reflection-in-action. Typically, they function as a central discussion point but were also used to house peripheral information, often physically relocated in close proximity to the students. The whiteboards, mixed with the use of the work on the walls, often led to frequent reflection: students would interrupt their current tasks to look and interact with the displayed work. Interestingly, the studio’s digital tools—a large wall-mounted touchscreen and horizontal touch table—were rarely used by the students, because they’re smaller and not as simple/easy to interact with.

**Multiple monitors.** The studio provided at least one desktop computer per student, but we observed that students brought their own laptops and plugged them into unused monitors (or they simply logged into a nearby desktop in addition to their laptop). This created unique group dynamics, but also allowed information to be kept on secondary screens. We found that this increased peripheral information and better supported reflection-in-action for a student’s current activities.

**Impromptu teaching.** One approach to teaching in the studio is the use of impromptu teaching, which is where a staff member teaches or demonstrates to a student, a group, or the entire room, at any time, based on his or her assessment of whether the students require it or not. One important part of the studio is its flexibility for teaching. It provides additional opportunities for the students to reflect on their understanding of a subject, but staff can also reflect on students’ progress and the course itself, providing impromptu teaching where necessary, for example, with demonstrations or short discussions.

**Peer critique.** Our studio environment nurtured feedback among students by encouraging critique from peers (a few examples, such as peer testing, are given later) and fostering a culture that openly supports critique. Through critiques, people can reflect on their work to either find alternatives or understand why their solution is optimal. This
forms a significant part of the studio culture.1

Reflection in Practice
At the end of the year, the students felt that they had learned and practiced a lot, with some believing that they had learned more than just the curriculum. It gratified us to hear that the students really enjoyed the course, with some liking it more than any other courses in their degree. Project size and complexity provided greater scope for reflective practice, but working in a studio better emphasized reflective practice. Several tasks were also explicitly designed into the course to elicit reflective thinking, some of which were used by the students later in the year without any prompting.

Task: peer testing. Our peer-testing task introduced the idea and benefits of getting their peers to test their code. Students had to reflect on their own testing practices and find better ways of identifying testable elements of code. We learned during our focus group that the students found this session to be the least useful and interesting, but they also admitted that this is potentially because they didn’t have a significant code base at the time that this session was planned, which prohibited the session from functioning optimally.

Task: peer code review. Our task related to code review had similar benefits to peer testing, but it looked more closely at code quality and design decisions. It didn’t have the same reaction as with peer testing session because it happened later in the year.

Task: requirement change. To simulate a real-world issue, we forced students to reflect on their work in its current state and reevaluate the project to address a new or edited requirement. Each group’s advisor formulated the new requirement on a per-project basis: it was estimated to take approximately a week’s worth of effort and got injected into the brief late in the project. Among other things, this encouraged reflection-on-action as the students reflected on their earlier design decisions and asked if they could have done something differently or they changed something to make addressing the requirement easier. In some instances it helped reinforce why good software engineering practices are important.

Task: Post-It questions. In some of the presentation sessions, students and staff wrote down positive points along with elements of concern on Post-It notes, which were shared and discussed after the presentations. One benefit to this is that the post-presentation sessions didn’t end up in lengthy Q&A exchanges, primarily because the thoughts and questions were documented and made visible on the walls over the rest of the course. This also helped overcome a few instances of awkwardness—regarding commenting on a peer’s work. On several occasions throughout the year, the students reflected back on the positive comments and concerns. We note, however, that this is a possible avenue to explore for scaling the studio.

S tudios aren’t limited to education: many elements of studio education work for practitioners, too. One obvious yet significant benefit is that they foster reflective practice.2 As you would imagine, education forms the foundation of how people will work in their future lives, but it can also form barriers that need to be overcome to be more in line with institutions’ expectations. If we want to better encourage reflective practice, we need to change the way software engineers are taught before their practice becomes too entrenched. Studios are just one approach that could achieve this. We believe that supporting and encouraging reflective practice in early career individuals is crucial as well.

To support a studio culture, we need to construct a culture where critique and questioning are socially acceptable. Critique and negativity aren’t synonyms! The idea of shying away from critique because of fears of negative feedback or of putting strain on social dynamics is fundamentally dangerous, and work could potentially suffer in the long run.

Don’t fear awkward questions. To foster reflective practice, start with constant questioning and go from there (more detailed elements of studio education that could be beneficial and help support reflection appear elsewhere8).

If you’re considering studios for education, here’s something to
remember: Schön states that in the early phase of an architectural studio, students experience mystery and confusion, which is “characteristic” of studios. This is normal and resonates with the paradox of learning to design, which can’t be taught but instead must be practiced. In a letter to Schön, Thomas Cowan distinguishes between education and training:

- “Education—the self-learning process.”
- “Training—what others make you do.”

The big question, then, is whether you want future colleagues educated or trained?

Acknowledgments

This work was undertaken as part of the postdisciplinary HighWire CDT programme at Lancaster University. HighWire is funded by the Digital Economy Programme, a Research Councils UK cross-council initiative led by the Engineering and Physical Sciences Research Council and contributed to by Arts and Humanities Research Council, Economic and Social Research Council, and the Medical Research Council.

References