Taking a Studio Course in Distributed Software Engineering from a Large Local Cohort to a Small Global Cohort

WILLIAM BILLINGSLEY, ROSEMARY TORBAY, and PETER R. FLETCHER,
University of New England, Australia
RICHARD N. THOMAS, JIM R. H. STEEL, and JÖRN GUY SÜß, The University of Queensland

One of the challenges of global software engineering courses is to bring the practices and experience of large geographically distributed teams into the local and time-limited environment of a classroom. Over the last 6 years, an on-campus studio course for software engineering has been developed at the University of Queensland (UQ) that places small teams of students on different features of a common product. This creates two layers of collaboration, as students work within their teams on individual features, and the teams must interoperate with many other teams on the common product. The class uses continuous integration practices and predominantly asynchronous communication channels (Slack and GitHub) to facilitate this collaboration. The original goal of this design was to ensure that students would authentically experience issues associated with realistically sized software projects, and learn to apply appropriate software engineering and collaboration practices to overcome them, in a course without significant extra staffing. Data from the development logs showed that most commits take place outside synchronous class hours, and the project operates as a temporally distributed team even though the students are geographically co-located. Since 2015, a course adapted from this format has also been taught at the University of New England (UNE), an Australian regional university that is also a longstanding provider of distance education. In this course, most students study online, and the class has to be able to work globally, because as well as students taking part from around Australia, there are also typically a small number of students taking part from overseas. Transferring the course to a smaller but predominantly online institution has allowed us to evaluate the distributed nature of the course, by considering what aspects of the course needed to change to support students who are geographically distributed, and comparing how the two cohorts behave. This has produced an overall course design, to teach professional distributed software engineering practices, that is adaptable from large classes to small, and from local to global.

CCS Concepts: • **Social and professional topics** → **Software engineering education**; • **Applied computing** → *Collaborative learning*; *Distance learning*;

Additional Key Words and Phrases: Global software engineering, studio pedagogies

This work is supported by the Australian Government through the Office for Learning and Teaching under grant SD15-5190.

Authors' addresses: W. Billingsley, School of Science and Technology, University of New England, Armidale, NSW 2350, Australia; email: wbilling@une.edu.au; R. Torbay, and P. R. Fletcher, School of Education, University of New England, Armidale, NSW 2350, Australia; emails: {rtorbay3, pfletch2}@une.edu.au; R. N. Thomas, School of Information Technology and Electrical Engineering, The University of Queensland, St Lucia, QLD 4072, Australia; email: richard.thomas@uq.edu.au; J. R. H. Steel, CSIRO, UQ Health Sciences Building, Herston, QLD 4029, Australia; email: jim.steel@csiro.au; J. G. Süß., Oracle Labs Australia, 340 Adelaide Street, Brisbane, QLD 4000, Australia; email: joern.guy.suess@oracle.com.

1946-6226/2019/01-ART13 $15.00
https://doi.org/10.1145/3218284

ACM Reference format:
William Billingsley, Rosemary Torbay, Peter R. Fletcher, Richard N. Thomas, Jim R. H. Steel, and Jörn Guy Süß. 2019. Taking a Studio Course in Distributed Software Engineering from a Large Local Cohort to a Small Global Cohort. *ACM Trans. Comput. Educ.* 19, 2, Article 13 (January 2019), 27 pages.
https://doi.org/10.1145/3218284

1 INTRODUCTION

Over the last 6 years, an on-campus studio course for software engineering has been taught at the University of Queensland (UQ) that places small teams of students on different features of a common class-wide product. We describe the course as *supercollaborative* because there are two layers of collaboration: students work within their teams on individual features and the teams must interoperate with many other teams on the wider product.

The course is based on studio pedagogies, which have a long history in design disciplines [45] and over the last 20 years have become increasingly valued for teaching software engineers to be reflective practitioners [19, 52, 68]. Studio courses can vary significantly, but they typically focus on collaborative and cooperative work on more authentic tasks, with feedback from instructors and peers given while work is in progress. This creates a learning environment centred around the learners' action and what they need to learn in order to accomplish their design goal [40]. Studio pedagogies are well suited to developing reflective practitioners who can grapple with realistic "messy" design problems [64, 65].

The original aim of our supercollaborative design was around general software engineering [5, 67], to ensure students authentically encounter the difficulties of scale that motivate many of the professional practices they were learning. We had observed that on smaller more traditional group projects, of perhaps four students, it was possible for students to produce programs that work without applying version control, continuous integration, or other practices that the course sought to teach. However, with a class-wide project where students could not individually know the whole code base, the software engineering tools and practices we were teaching became a necessity. A noticeable educational advantage inherent in a supercollaborative design is that as the class was not fragmented across many projects, instead students are engaged in a collective endeavour, it was easier for the teaching staff and the students to follow the development of the class as a whole. For the students particularly, this provided richer opportunities for learning from the experiences and practices of other groups in the class. Not only would students observe when another team breaks the build or encounters "merge hell," but they would also interact with other groups' designs and code across the project.

Although our reason for increasing the scale of collaboration in the UQ course was to coerce students to apply general software engineering practices, we have observed that it also scales the collaboration beyond a size that could be coordinated in regular class time alone. One of the constraints on the course design was that it should only use scheduled teaching spaces rather than have its own dedicated full-time studio space [67]. Because the teaching spaces are only available for short periods during the week, most development work and communication takes place outside the classroom using continuous integration practices and predominantly asynchronous communication channels (Slack and GitHub, using an issue tracker and wiki for feature coordination). For instance, data from the development logs, as illustrated in Figure 1, shows that most commits are widely distributed throughout the week, even though the studio course represents only one-quarter of a student's full-time load for a semester.

This combination of short periods of synchronous communication (scheduled studio time) with extensive asynchronous collaboration throughout the remainder of the week has some

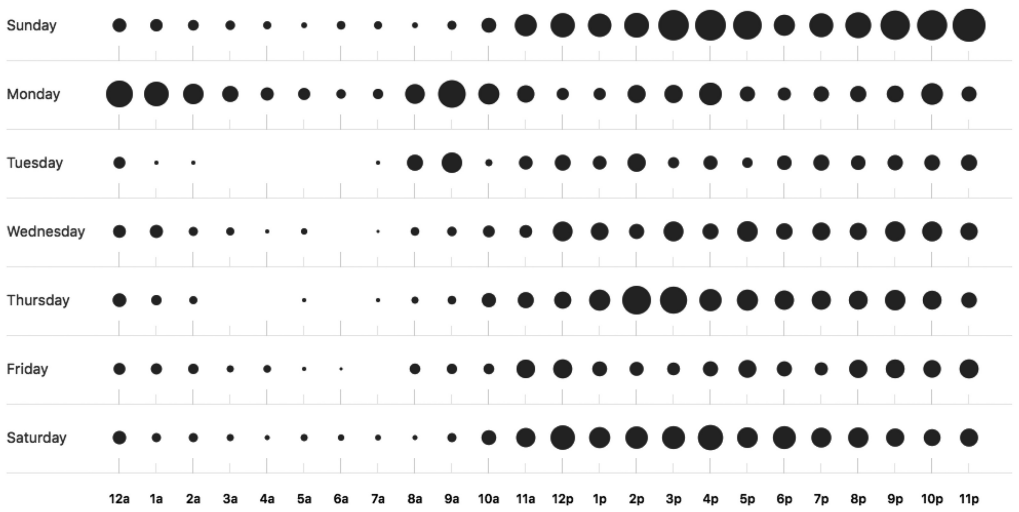

Fig. 1. Data from the version control system showed that most of the work was distributed outside class hours, when the class would have to operate as a distributed team. A visualisation of 2014 data, the year before the off-campus course ran, is shown.

correspondence with the *balanced synchronicity* communication model in Britto et al.'s extended global software engineering taxonomy [8]. This is not an unexpected result—the course design intentionally mimics the practices of distributed software development—but the data does indicate that the project in this regard behaves as a temporally distributed team even though the students are geographically co-located. There are, however, very significant differences from global software engineering projects:- language and cultural dimensions are not captured, and the temporal distance students encounter is of a different kind than would normally be encountered in a global project. Whereas taxonomies such as Šmite et al. [66] classify temporal distance in terms of the timezones participants must operate within, in this class the temporal distance is created by the varying schedules, habits, and other time constraints of so many students.

Since 2015, a course adapted from this format has also been taught at the University of New England (UNE), an Australian regional university that is also a longstanding provider of distance education. At UNE, most students study online, and although the class size is smaller (approximately 50 students) it has to work globally, because as well as students taking part from around Australia, there are typically also a small number of students taking part from overseas. As many off-campus students are in full-time employment, the class also has to be able to operate asynchronously. Unlike popular Massively Open Online Courses (MOOCs), it cannot rely on having a critical mass of students available for synchronous online sessions, and as a compulsory course within a degree, cannot disadvantage students who cannot make regular class hours.

Taking the course from a large on-campus cohort to a smaller distributed cohort has allowed us to further examine the hypothesis that the course design promotes and practices distributed development. For the UNE geographically distributed variant of the class, comparatively few adaptations had to be made to the course design to allow it to run. Principally, the critique process, whereby teams present their work during development and other students are asked to offer a critique and advice, was adapted to work via video submissions and replies. Analysis of the geographically distributed class's development also allows us to discover what qualitative differences there are between how a supercollaborative on-campus project behaves, and how a genuinely distributed class behaves when developing software.

Our research takes an Action Research approach [7, 13, 41]. Over a number of years, a group of researchers and practitioners have sought to bring together theory, action, reflection, and practice to improve software engineering studio education. We regard this approach as particularly appropriate for a course using studio pedagogies, which are themselves grounded in theories of reflective practice [65] and situated learning [45].

The design of the first two iterations of the on-campus supercollaborative studio have already been presented [5, 67]. The focus of this article is on its adaptation and transfer to run in a predominantly distance education mode. In order to examine reflectively, from the data that the course produces, how on-campus supercollaboration reflects the practices of distributed teams of students, we focus this article on the following research questions:

- —What changes would enable a supercollaborative course to run effectively for a class with a mixed cohort of geographically distributed and on-campus students?
- —Are there apparent qualitative differences between the behaviour of an on-campus supercollaborative class, and a geographically distributed supercollaborative class?
- —Phenomenologically, how do students critique each other's work when working asynchronously over video as they cannot meet on campus?

To situate these questions, in the next section we will describe how the on-campus course has evolved since it was first presented. For example, the critique process that we adapted and examined for UNE's distributed class was not included in the first two iterations at UQ.

2 RELATED WORK

2.1 Design Studios in Software Engineering

The on-campus course was designed at the University of Queensland, which has a long history of bringing studio practices into software engineering [19]. As this is the lineage of the course design, it is worth explaining the history and pedagogy of studio.

As a teaching environment, studio has a history that stretches back to schools such as the *École des Beaux Arts* in the 19th century. However, most modern studio pedagogies take their inspiration from current educational practices in architecture, design, and planning schools. Students are given realistic multi-faceted design problems, and a shared learning space with other students working on their own design projects. Work in the studio is seen as cumulative, with frequent critiques during the course of a project through "pin ups," where work is pinned to the wall for the class collectively to critique, and desk critiques between the instructor and participants. These studio settings have long dominated architecture and planning education [20, 45], because they are well suited to the ill-defined and far-reaching problems that design challenges encompass. Levy [40] observed that the studio is the environment in which all aspects of architectural skills can be learned and practiced together, and that the design goal drives students to learn what they need in order to complete it. Schön [64, 65] focused on the studio as a means to develop "reflective practitioners," who can leave the high ground of theoretically solvable problems, and enter the "swampy lowlands" where lie the messy, confusing problems that are of greatest human concern.

Recognising that software development is also a field of messy design problems with no perfect solution [9], Carnegie Mellon University proposed the first software development studio course [68, 69] as a significant experience that would promote reflective practice and get students beyond "throwaway mode" in their development. Soon after, many more computer science courses took up and adapted studio pedagogies [3, 27, 31–33, 38, 62].

Although not labelled as a studio, Fox and Patterson's Software as a Service course [26], which asks students to develop software in an agile manner with a customer, also has some characteristics

common to software studio courses, and as its teaching material has been adapted to deliver a popular MOOC and textbook, is possibly the most widely known. McDermott et al. [46] examined the link between projects and reflective practice by asking students in a global software engineering education project course to write reflections on cultural differences and on their use of communication technology. They found the reflections on cultural differences showed significant depth of reflection, although those on the technology were shallower, perhaps because communication technology is so ubiquitous as to be almost invisible.

In the direct lineage of our research, the University of Queensland's Bachelor of Information Environments degree [19] was designed extensively around studio practice, including a studio in every semester of the students' studies. Although this degree, which took place at a satellite campus, has since closed, studio pedagogies have been incorporated into the university's information technology degrees on its main campus.

As different universities have adapted studio for their software development courses, they have diverged significantly from its origins in design education, and there are now a wide variety of approaches. For example, Tomayko's original software studio [68] and Nurkkala and Brandle [52] stress the importance of a real customer, even though studios in traditional design disciplines frequently use just a simulated brief [17]. In response to how divergent studios had become, Bull et al. [11] conducted a series of interviews with designers, architects, and artists, to establish a model of which characteristics of a studio are considered important to traditional design disciplines, as a way of "evaluating software studios to determine if they are in fact a studio." Although their eventual model is not prescriptive, they came up with a set of 10 dimensions on which studios could be plotted, including the sense of a shared space (the studio), the "ownership" of the space by the students, the practice of critique, and awareness of each others' work.

When we were first designing the supercollaborative studio course [67], we found that many of the existing courses were difficult to apply to large-scale undergraduate education. Early software studios were frequently post-graduate [38, 68], or were highly selective in the students they admitted [62]. Some [3] relied on the close involvement of customers and technical experts with the teams. Others [32] used studio pedagogies only for short assignments of a few weeks within a class. Others [12] placed students into relatively small groups on independent projects; this, we had observed, meant that students could produce working software without needing the software engineering skills we were seeking to teach, and so software engineering aspects, such as good version control practices and testing, could be seen by students as busy work or only applied after-the-fact.

2.2 Online Studios and Teamwork

Although studio approaches to software engineering have typically only been offered on campus, design schools have taken studio pedagogies online for many years [35, 37]. These use online collaboration tools, virtual worlds, social networks, or other computer-supported cooperative work environments, to enable creative collaboration; the virtual studio space afforded by the tooling stands in place of a common physical space [17, 63]. This is not only motivated to enable remote students to engage in studio, but also because professional design work itself is now often computer-mediated, for example through clients submitting design briefs to online design networks and "crowdsourcing" environments [55]. In their Human-Computer Interaction studio course, Koutsabasis and Vosinakis situated studio critiques within a virtual world for on-campus students, linking studio practice to one of the environments of interest for the subject matter [34]. In our case, teaching distributed software engineering, the move to teach studio online similarly aligns with professional practice and the subject matter of the course; several tools that are widely used in global software engineering [57] become principal components of the virtual studio space.

However, technology-mediated studios do have their drawbacks. For example, in Bull's interviews with studio participants in traditional design disciplines [11], he found that the use of technology was discouraged as it tended to make work less visible to other occupants of the studio.

Some design studio courses are offered virtually because the institution specialises in distance education. Lloyd [44] describes an introductory design thinking course at The Open University, which incorporates a bespoke social platform where students upload work for discussion by their peers. Among the MOOC providers, NovoEd (formerly Venture Lab) [61] has specialised in providing an online social learning environment. This was created from the need to support peer learning in an entrepreneurship course, and has since been used in a number of design-based courses, including by the design organisation IDEO.org [1]. Early courses on the platform included a lightweight critique process: Students would be asked to review work that had been uploaded by peers in the course, using a simple rubric laid out as a grid of text boxes. This lightweight critique process was one of the processes we considered when developing the three-stage critique process in the third iteration of our on-campus course.

MOOCs are now a very common form of remote learning, and there is increasing interest among researchers in how to promote collaboration in MOOC environments [2, 72, 73]. This has also included investigating the benefits of synchronous collaboration among MOOC students [23, 42]. However, it is important to note that there are significant contextual differences between distance education within a traditional university degree and MOOCs. For example, MOOCs are often cited as having very large numbers but tolerating attrition rates around 90% [53]. The low rate of participation extends to teamwork in collaborative MOOCs; for example, less than 10% of students in the NovoEd classes that Wen et al. [72] examined joined a team. This means that MOOCs are an environment that can take advantage of very large numbers to find pools of students who are available for synchronous activity at the same time, and that can tolerate a selection effect whereby students who find they cannot take core course activities simply do not participate. In our regional university distance education setting, the numbers of students is much lower, meaning fewer distance education students will find their availability for synchronous work coincides with other potential teammates, and as we need to progress students through their degrees, the tolerance for attrition within core courses is also very much lower.

2.3 Global Software Engineering in the Classroom

As distributed and global software engineering has become more prevalent in industry, many researchers have attempted to bring the challenges of geographically distributed teams into the local and time-limited environment of a classroom [24, 48].

One common approach has been to set up cross-institutional collaborations, whereby student teams from universities in different countries work together on a software project [16, 18, 36, 47, 54]. Each of the local teams meets together physically on a campus, but is geographically distant from their collaborators at the other institution. However, this requires coordination between universities and presents logistical issues such as whether the term dates at the institutions sufficiently coincide. Although these courses implement global software engineering projects, the educational setting can still alter the challenge. For example, Richardson et al. [59] note that because the participants are students and have other coursework to balance, they face additional scheduling challenges and competing draws on their time that a full-time professional development team might not encounter. McDermott et al. [46] also notes that in projects that engage with an external client, the clients' other obligations can limit their availability, making access to them problematic.

Another approach has been to run short simulations of global development on campus. For example, through virtual environments that replicate challenges of communication [49] or

requirements elicitation [60], or by dividing the class a into a small number of teams, each representing a different site, and regulating when teams can communicate with which other teams [39, 43]. Some other courses isolate particular practices that they ask students to perform with remote peers. Murphy et al. [50] introduced a distance education course that asked students to employ Extreme Programming (XP) practices and focused particularly on virtual pair programming, finding amongst other difficulties that it was difficult for non-co-located students to schedule synchronous sessions.

The approach we have effectively taken with the on-campus course—scaling the collaboration beyond what can be coordinated in scheduled class time—is, to the best of our knowledge, unique.

Noll et al. [51] identified three kinds of barrier to collaboration in global software development: geographic distance, temporal distance, and cultural distance. Of these, Clear et al. [15] found temporal distance challenges to be most prevalent in research in global software engineering education. They also found that there was extensive advice in the literature that instructors should become intimately involved with the teams and projects, while being flexible and anticipating change. This echoes the instructors' role in studios, where they constantly guide and advise students in their work, rather than simply delivering instruction. On teamwork, they found students needed to put a great deal of effort into establishing synergy with remote teams, and that this did not always occur. Gotel et al., [30], Favela [22], and Peña-Mora et al. [56] in various ways suggest that coming to a shared project vision is important for promoting synergy.

Translating our supercollaborative studio to a distance education environment inherently introduces two of these barriers: geographical distance as the students are remote from each other, and temporal distance as there are no common scheduled working hours for remote students studying asynchronously. Although our hypothesis in this article is that for on-campus students, scaling up the level of collaboration required is sufficient to cause the class to take on some characteristics of a distributed team.

Cultural distance is not directly addressed; however, Australian university courses typically include around 25% international students (albeit lower in computer science at UNE), so some cross-cultural collaboration is likely to be present.

3 ON-CAMPUS COURSE

3.1 Initial Description of the Course

The first two iterations of the on-campus course have already been described [5, 67], and are summarised for context here. The course is designed as a compulsory class in software engineering, that sits at second year in the programme for most students. Students enter the course already knowing how to write small programs in Java, but the course then seeks to teach them the professional practices and tools of how large and often distributed teams build larger software products together.

The class is structured around a class-wide project that begins at week three, so that we can teach the basics of build systems, version control, and architecture patterns first. The students are asked to form "feature teams" of approximately four students (eventual group size varies with attrition), who will each design and develop a feature for a common product.

For the first two iterations of the class, we asked students to add features to an extensive pre-existing code base, chosen so it would afford opportunities for refactoring and working with legacy code. We saw this as an important skill, as we expected the students during their careers would spend more time working with existing code bases than starting greenfield projects, and other researchers had also found that this is a skill that industry demands [25]. These first two iterations of the UQ course ran without a formal peer critique process; this was added in the third year

of the course, at a time when UQ's main information technology degree was being adjusted to incorporate some of the studio approaches from its earlier Bachelor of Information Environments programme [19].

The project infrastructure uses a continuous deployment tool chain, released progressively through term. This includes a build system, version control system, and collaboration site, an artefact repository from which the build system will fetch libraries, a recommended development environment and debugger, a continuous integration server, and a dashboard for software quality and project health metrics. Teams' user stories are managed as issues in the ticketing system, and teams also maintain a wiki page for their documentation. In the initial runs of the course, no common out-of-class chat system was provided, as students tended to self-organise this on Facebook.

The typical class enrollment was 70 students, with 19 features being developed on the common legacy base. In the first year of the course, these features were chosen by staff so that they would overlap and ensure teams would need to communicate with each other. In the second iteration of the course, student teams were given free reign to propose their own features, with the result that class communication spiked earlier as teams needed to negotiate with each other to avoid duplication but ensure they could work with other teams' features [5]. The layered collaboration meant that each student team was working with a code base that was in flux. This meant they needed to employ their professional skills such as testing, debugging, and continuous integration. As the number of teams on the project was large, it also ensured that in-person communication would not be sufficient. Interteam communication needed to occur outside class time, when students do not have an allocated space, which allowed the course to model the practice of distributed teams.

3.2 Evolution of the On-Campus Course

Since the on-campus course was described [5, 67], it has undergone some significant evolution. In 2013, UQ brought studio pedagogies more thoroughly into the information technology degree on its main campus. Our supercollaborative studio became the second studio course in a sequence of four, and the number of students entering the course increased significantly, to approximately 140 in 2013 and 200 in 2014. We also made a number of changes to strengthen the studio aspects of the course; particularly, to give students greater ownership of the product, to promote a sense of shared vision, and to introduce a critique process.

Whereas we had theorised that using a large legacy code base would encourage students to engage in refactoring, in practice this did not occur. Instead, we found that students mimicked the design of the existing code. For instance, if there was a "God object" anti-pattern in the existing code base, teams would weave yet more functionality into it; in one case growing a single Java class from 1,679 lines of code to 3,459 lines across 344 commits by 53 committers. This may be due to a "tragedy of the commons," where no single team would benefit enough from the refactoring for the difficulty (and risk of breaking things) to be worthwhile to them. Or it could also be explained simply as students taking poor code supplied by academics and professionals as an example to be copied rather than as a legacy to be improved. This led to a revision of providing a smaller initial code base, allowing students to start implementation immediately rather than wait while they investigate an architecture or framework to tackle the solution. This gives students greater control of the overall design, and means less time is required to know the code base before they can begin their project. However, it also implies that there will not usually be a single coherent product at the end of term. For example, although features will work together, there may be many missing features—things that it would make sense to implement but that no team has chosen to take on. Also, well-designed features from high-performing teams sometimes need to integrate with buggy features from less active students. The project becomes a challenge for each team to

develop a working and coherent feature while working with other teams' code, rather than for the class to develop a complete working product.

Also in 2013, to strengthen the studio nature of the course, a critique process was introduced. As the class size had doubled, the class would not fit in a single tutorial session in the allocated room and there would not be time for the whole class to focus on each groups' work as might happen in a traditional design studio pin-up critique. We introduced a critique process where the teams present their work in progress in class and the critique is then conducted asynchronously out of class. Each student was randomly allocated a number of groups in the same tutorial session to critique, and for each group was asked to submit their advice in an online questionnaire. This advice was then automatically anonymised and shown to the group under critique, who would indicate whether they found the advice to be specific, constructive, actionable, and helpful. A small open source system, Assessory, was written to handle this process, as the review of students' critiques could produce more than 3,000 forms across the semester [6]. Checkpoints were placed into the course, at which students would present their work in progress for critique by their peers and for marking by their assessors.

In 2014, when the class size had grown to 200 students, we observed that there were too many groups on the project for students to track, and that the project had become incohesive and the process chaotic. This was observed informally; early in the course, interacting with students in the tutorial sessions, we observed that more students appeared to be expressing concerns about who was "in charge" of the project, and some design disagreements between teams took longer to resolve. The student feedback in 2014 also fell despite there being few other changes being made to the course.

For the 2015 class, three changes were made to address this. The class was split by tutorial group, so that instead of there being a single class-wide project, there was one project in each tutorial group. This returned the project size to something closer to our original design, with 30–60 active committers on each project (not every student within a team necessarily commits code). The second change was the addition of Slack as an asynchronous communication mechanism for the class, replacing the students' previous use of a Facebook group. The third change was the introduction of a design committee; one member of each group would be elected to the design committee that had responsibility for the overall product vision of that tutorial group's game. This has similarities with the findings of Clear et al. [15] that a shared product vision is important to produce synergy between teams.

The on-campus studio course remains a core part of the curriculum at UQ. Recent changes since 2016 have included expanding the role of an individual portfolio of each students' contribution to the project, and examining ways to make it easier for students to familiarise themselves with the project and how to contribute to it.

4 ADAPTATION TO A DISTRIBUTED CLASS

4.1 Context

UNE is a regional university that is also a longstanding provider of distance education. Student demographics and enrollment vary from year to year, but in 2015 across the undergraduate computer science and postgraduate information technology degrees, 78% of students were registered as distance education students, and 57% as part-time. 44% of students were located in major cities around Australia (including 16% in Sydney and 5% in Brisbane) and 45% were located in regional towns and cities. More than half of the distance education students reside in locations with a university that teaches computer science or information technology, suggesting that they do not study online because they are distant, but for other reasons such as to fit in with their schedules. Most

students are within Australia (including 8% international students who study on campus), but each year we do have a small number of students who are studying online from overseas, and have to be able to cater to them.

In 2014, the university was examining how to redesign its computer science degree. As well as updating and focusing its programme, a significant concern was how professional skills, such as teamwork and interpersonal communication skills, could be practiced by students who study remotely throughout their degree. As the data from the UQ course had shown that only 18% of commits occurred during scheduled class time [5], and we had previously modelled how a super-collaborative MOOC might work [6], we decided that it would be viable to adapt the studio course to run for a mixed on-campus and distance education class. The course first ran in 2015, ahead of a new degree design in 2016.

The studio course is a core course at second year level for undergraduates in computer science, who have previously taken two first year programming courses, and is also available to undergraduates in science and education. It is also a compulsory course in the university's conversion masters degrees in information technology, for students whose bachelor degree was in a different discipline. These students typically have less programming experience than the undergraduates. Typical enrollment is between 40 and 70 students, which is somewhat smaller than even the first iteration of the course at UQ. In 2015, 57% of students were studying via distance education, 34% were studying part-time, and 27% were enrolled in degrees other than the Bachelor of Computer Science. The term length is 2 weeks shorter than at the UQ, with 13 weeks prior to the exam period (14 in 2015), including a 2-week nonteaching period in the middle of trimester.

Students sometimes change their mode of study during the course. For example, one student in 2016 began the course on campus before moving to Europe and studying remotely 1 month into term. Another in 2015 studied online, but from a town close enough that they were able to travel to campus for a small number of the studio sessions. Another took the course from Asia in 2015, but has now moved to Australia to complete their studies in other courses on campus.

The distance education component of the class means the studio has to be a digital space—a set of collaboration tools, methods, and critique practices—rather than a physical space. However, this also means the studio has the additional role of trying to enhance the sense of community among the class. Whereas on-campus students may share cafés, libraries, and study spaces, the distance education students lack an out-of-class community, and as they are dispersed across so many locations around Australia and (sometimes) overseas, some students entering the course might not have seen or heard any of their classmates before.

4.2 Initial Changes

The course at UNE was first taught in 2015, and was kept initially similar to the 2014 UQ course to enable comparison. This included using the same starting code base—a physics-enabled open world game using a top-down view, with a client and server communicating over Transmission Control Protocol (TCP). A minimal set of changes were made, as we considered the question of how the course needed to be altered to make it work for a distributed team.

The most pressing need was to add an asynchronous chat mechanism. While in 2014, UQ students had used a self-organised Facebook group, we were aware that not all the class had joined the group, and some students might be reticent to use Facebook for university purposes. We added a Slack group for the course in the third week of term. Coincidentally, the UQ course also added Slack in 2015.

Some immediate changes needed to be made to the critique process. Whereas in the on-campus course, work-in-progress could be presented live to peers in a studio session, it was not going to be possible for distance education students to present synchronously to the whole class. Instead,

the presentations were recorded as video. For the 2015 class, the critique stage was kept using text as at UQ; however, the allocation mechanism needed to change. At UQ, students needed to know which group they would be critiquing before the presentations, so they would know to pay careful attention. At UNE, the videos would not all be posted at the same time; some students would inevitably be late submitting. The allocation mechanism picks the least-critiqued videos that are available for review. This gives a soft incentive to post on time, as the pool of potential critics to receive advice from shrinks as students open the critique task and receive their allocation. Late-submitted videos are therefore likely to receive fewer critiques, and from students who are themselves late submitters.

4.3 Critique Process Over Asynchronous Video

In 2016, we modified the critique process so that students would record their critiques (as well as their presentations) as videos. An illustration of this process is shown in Figure 2.

This adaptation was made to allow students to present critiques more richly, for example, including demonstrations of an issue, sketching alternative designs, using document-camera interactions to focus attention, and being able to use deixis and point at things rather than having always to describe them in text. We regarded this as important, as the class's existing collaboration mechanisms are dominated by text interactions. For example, the wiki, the issue trackers, the source code repository, and the forums all use text as their dominant medium for communication.

We conducted a pilot test of the video-based critique process in an Interaction Design course the previous trimester [4], and observed instances where a student would screen-record the playback of the presentation video, and scrub to points of interest to provide a commentary. This suggested there may be interesting interactions between how presentations are conducted and how critiques are conducted, as they would now be using the same delivery medium.

We also hope that this will provide an additional benefit of improving the sense of community in the class, as every student's voice will be heard by other students in the class, rather than only those students who make their team's video presentation. Usually, however, students will not see each other's faces as students are encouraged when the task is introduced to place subjects of interest in the frame, rather than record a talking head.

The Assessory open source critique tool was modified to accept links to videos in the cloud, and automatically display the video player. The player is shown to the person posting the video so they receive immediate feedback that they have posted the URL correctly, as well as to the students who need to watch the video. (For the critique stage, this is the students who have been allocated to review that presentation; for the critique review stage it is the students in the group that are the subject of the critique.) Assessory supports the Learning Tools Interoperability (LTI) standard, so that accessing the task is just a matter of clicking on a link from the class learning management system.

Currently, YouTube and Kaltura are supported as cloud video providers. YouTube has the advantage that many recording tools, such as mobile phone cameras, already include functionality to compress and share a video to YouTube, saving students the effort of installing specialised tools or the inconvenience of waiting for very long uploads. Kaltura has the advantage that it allows students to upload video using their existing university account, so they do not need to register for YouTube and accept any associated terms and conditions.

4.4 Differences Between the 2015 and 2016 Iterations

As data from the 2015 and 2016 iterations of the UNE course are described, we describe here some of the contextual differences.

In 2016, the university shortened the length of its trimesters, from an elapsed period of 14 weeks from the start of lectures to the start of the exam period, to 13. In each case, this includes a 2-week

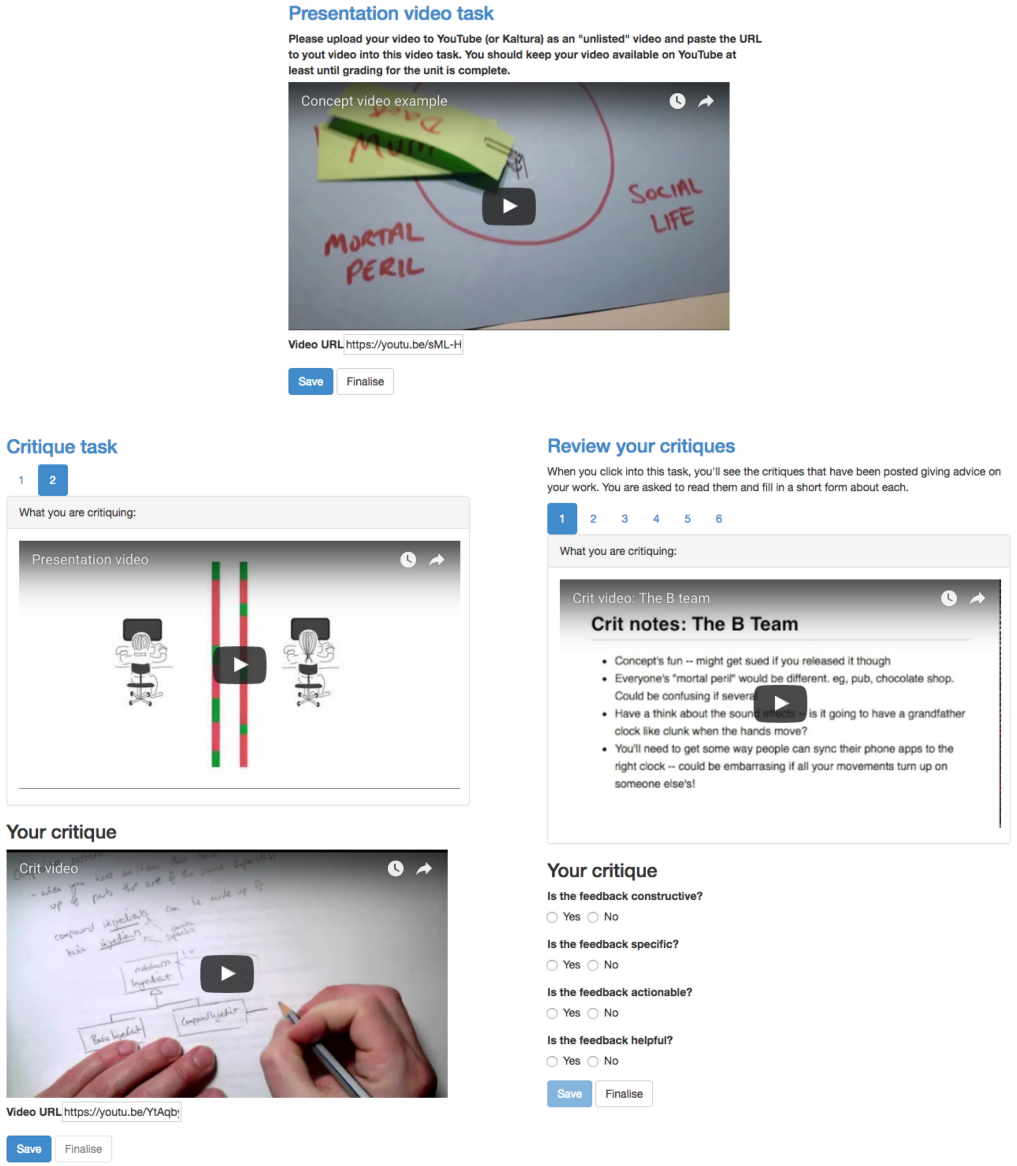

Fig. 2. The critique process works via video, giving flexibility in what students choose to show. It also enables students to hear each other's voices, although not normally to see each other's faces. *Top:* Groups upload their work-in-progress presentations as videos. *Left:* In the critique task, each student in the class is dynamically allocated a number of presentation videos to watch and reply to. The critiques are also recorded as videos. *Right:* When reviewing their critiques, students are asked to indicate whether each critique is constructive, specific, actionable, and helpful. For confidentiality purposes, the screenshots in this figure have been generated using sample videos rather than student submissions.

non-teaching period in the middle of the trimester, which the university uses to run "intenstive schools" on campus for distance education students in lab-based courses such as some sciences. Internal data within the university shows that this shortening of the term length caused a small but measurable increase in attrition and a reduction in the number of courses students take in a single trimester across the university. It also created an increased sense of urgency.

With the degree, the updated programme was rolled out in 2016, with a change to what majors are offered and what courses are required. Every course in the degree was in some way different to the 2014 offering. This caused some transience in enrollment patterns, as students who were part-way through their degrees had individual programmes mapped for them either to complete their existing degree requirements with the new courses, or to transfer to the new degree rules with credit for their previous courses.

Within the studio course, whereas in 2015 the subject matter for the studio was an open world game (as it had been in the 2014 UQ course), in 2016 we attempted a non-game topic. The project was called Sense-O-Matic, and asked students to create a suite of data analysis and visualisation tools. The intention was to give a project theme that connected to precision agriculture, a field that UNE is well known for in Australia, and which elsewhere has recently seen an upsurge in start-up activity and venture capital investment.

The assessment structure was also altered at a surface level. In 2015, the two halves of the project were phrased as two distinct assessment phases that follow on from each other. Combined with the mark scheme, this had the consequence that there was a set portion of marks for committing regularly before the mid-trimester checkpoint, and a set portion of marks for committing regularly after this point. In 2016, this was softened to be a critique point within a single trimester-long assessment item. The aim was to take into account the wide variation in experience and ability of the students at the start of the project, and leave the allocation of marks for production either side of the critique boundary fluid. This was to avoid accidentally penalising less experienced groups who might take more time to come to grips with the project, and so might have more sparse commit records in the first half of the trimester than the second.

The course continues to be a core part of the curriculum at UNE, and from 2017 is also supported by the introduction of a first year studio including smaller collaborative assignments, so that students are introduced to collaborative work from the beginning of their studies. The course also continues to evolve; for example, in 2017 we introduced a design committee based on its success at UQ in 2015. For future iterations, we are looking at changes in how we deliver teaching content to the mixed on-campus and online cohort, how to improve our use of student-facing learning analytics, and how we build students' skills in presentation and critique. We are also constantly examining how to improve the hectic period of familiarisation at the start of the project, when students have to come to grips with the project aims and a new code base, whilst beginning to use tools they have only just been introduced to.

5 DATA AND OBSERVATIONS

5.1 Commit Distribution

The commit punchcard for the 2015 class, visualising when commits occur throughout the week, is shown in Figure 3. The class size is very much smaller than in the equivalent chart for the UQ course in 2014, shown in Figure 1. Consequently, periods of peak activity amongst groups stand out more within the chart. For comparison, the punchcard visualisation from one of the comparatively sized projects at UQ in 2016 is shown in Figure 4. Commits in the UNE predominantly distance education class are distributed extensively throughout the week, but not noticeably more so than in one of UQ's equivalently sized projects. This suggests that whether a supercollaborative studio

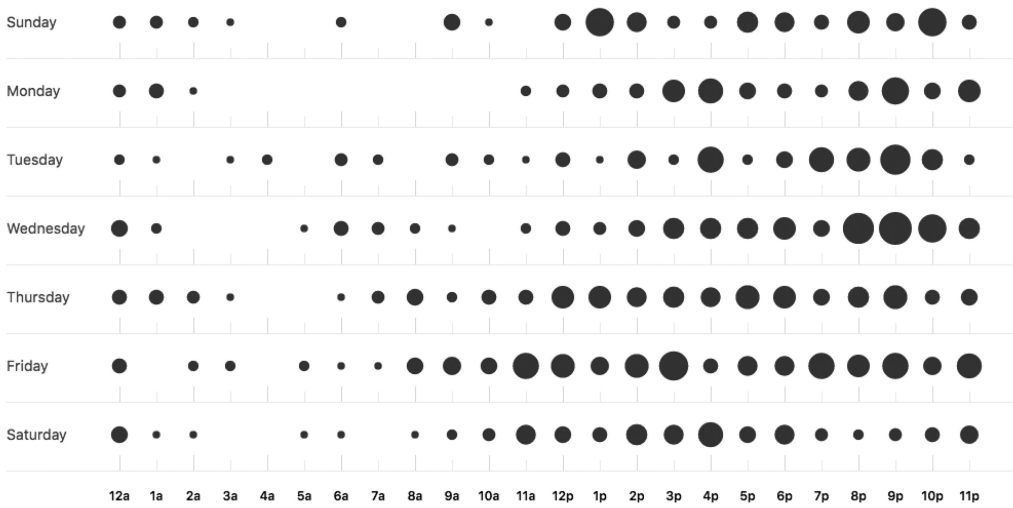

Fig. 3. Commit punchcard from the 2015 course at UNE, which used the same project topic as the 2014 UQ course. The class size is much smaller, with 35 active committers compared to 130 at UQ.

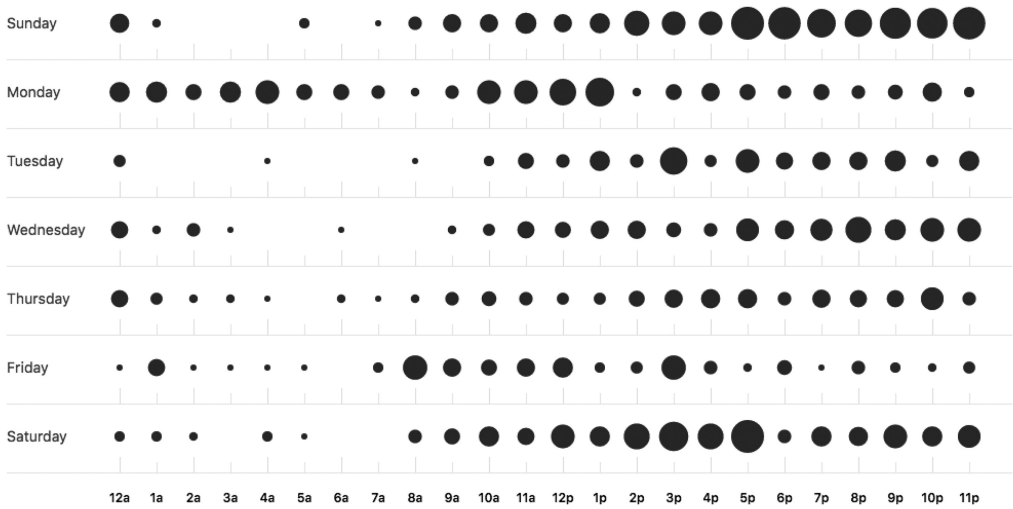

Fig. 4. Commit punchcard from one of the comparatively sized projects at UQ in 2016.

class runs entirely on campus or in distance education does not make a qualitative difference to the temporal distance between the groups; they still operate with far greater variability in their working times than their time on the project (one-quarter of a full-time load) would suggest.

Figure 5 shows the total commits per week and the merge commits per week of the UNE courses in 2015 and 2016. The tendency for work to increase as a checkpoint deadline approaches is noticeable in both these charts, and is noticeably more severe in the 2015 version of the course, where the two halves of the project were phrased as separate assessment items. Our previous presentation of the on-campus course at UQ [5, 67] showed similar spikes in activity near deadlines. This suggests that whether a supercollaborative project is run on campus or distributed, it will still be drawn to follow the cadence of university course with marking deadlines. The total number of commits is

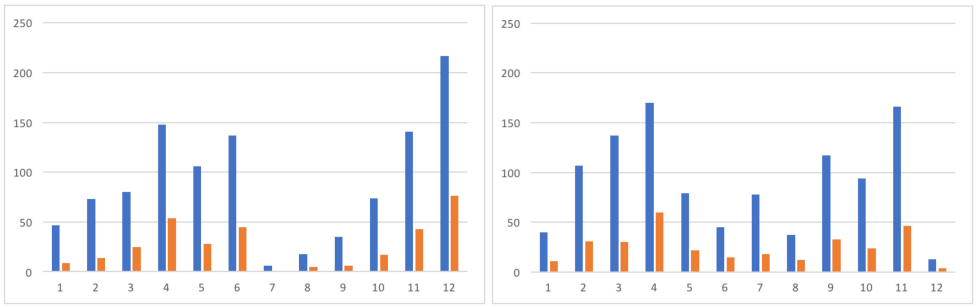

Fig. 5. The total number of commits per week and the number of merge commits per week across the project. *Left:* 2015 data. *Right:* 2016 data. Weeks 6 and 7 are the non-teaching period in both charts.

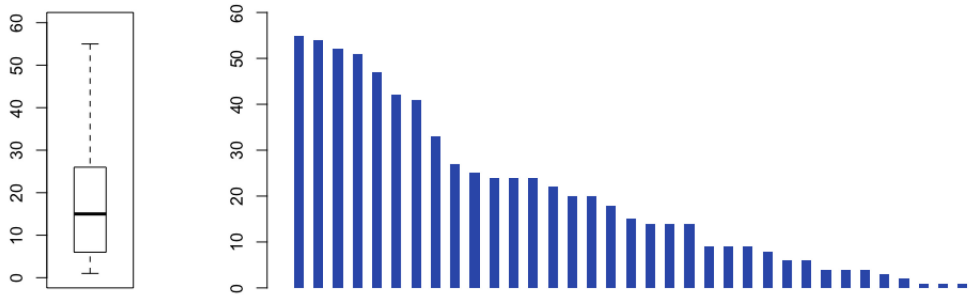

Fig. 6. Commits per committing student at UNE in 2015. (M = 20.1, SD = 17.0).

very similar across both years of the course: 1,157 in 2015, of which 809 were non-merge commits, and 1,160 in 2016, of which 832 were non-merge commits.

The number of merge commits per week follows a similar trajectory as the total number of commits per week. This indicates that despite being temporally and geographically distant, students were succeeding in integrating their work with one another; there is no indication of the teams diverging from one another during term and having a sudden rush of merge activity at the end. More than 1,000 commits were made across the project each year, with only 58 not merged into the master branch by the end of the trimester in 2015, and 52 in 2016.

The total number of commits per user, anonymised and shown in descending order, is given in Figure 6. This does show a wide variability; activity is not evenly distributed across the class. Two effects are likely to be conflated within this chart. Partly, this can be explained by variability in skill level and engagement within a class, but also partly by students specialising within their teams; for example, the student presenting the team's work-in-progress videos is not always the highest committer within the group. Although the lecturer, who produced the initial code base, is excluded from this list, it is perhaps worth stating that if the committers to the project are rank-ordered by number of commits, he is 25th in the list. In other words, more than half the class made more commits to the project than the staff made in creating the starting code base.

In this measure, we prefer to use the number of commits, rather than the size of the commits, as the course encourages continuous integration practices. Larger, less frequent commits would make a similar contribution in terms of code, but would give the rest of the class less opportunity to work with that code while it is under development. The initial code base is released to the class as several small coherent commits, in order to model this practice.

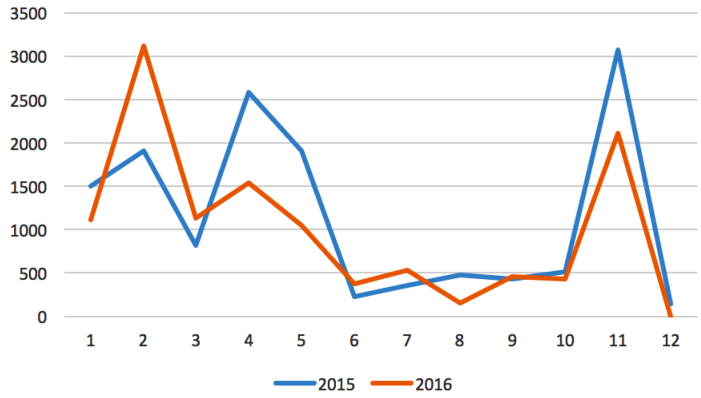

Fig. 7. Messages per week on Slack after the Slack group was opened in 2015 and 2016.

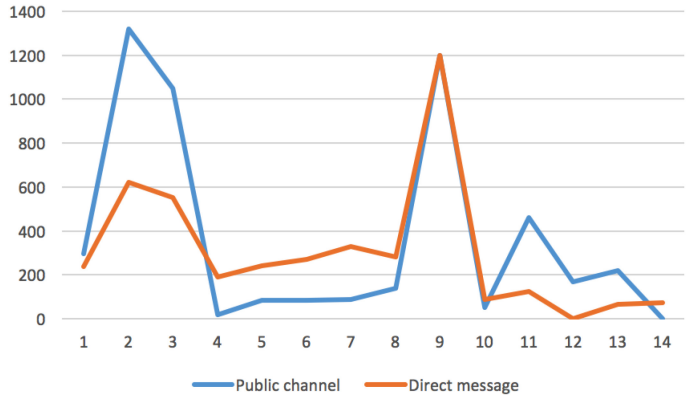

Fig. 8. Messages on public channels and direct messages in 2015, after the Slack group was opened.

5.2 Slack Activity Patterns

Although code contributions are a useful indicator of whether the class is succeeding in combining its work, they are not the only form of activity in a project. Clear et al. [15] found that it requires a large amount of communication and coordination effort to engage distant teams, and that clear and consistent communication is important to building synergy. In this section, we consider the communication data that comes from the Slack platform, which is by far the highest volume of communication in our course, in order to examine when chat communication is occurring, and how participation in chat is distributed across the class.

Figure 7 charts the volume of messages per week across the course after the Slack channel is opened. Noticeable increases in message flow can be seen soon after it opens, and approaching both checkpoints. The peak near the first checkpoint is substantially lower in 2016 than in 2015. Again, this suggest that the project's work patterns are significantly deadline-driven, and that the hard separation of marks for work before and after the mid-trimester checkpoint in 2015 emphasised this effect. Figure 8 charts the numbers of messages on Slack that were in public channels versus those that are direct messages. The peaks in message flow are visible in both categories, but there is visibly greater variation in the public channel.

Although there is great variation in the amount that is posted on the public channel throughout the term, the readership is more consistent. Figure 9, taken from Slack's statistics feature, shows

Fig. 9. The small version of Slack's analytics chart of active readers across the project in 2016. The peaks and troughs in workload are less apparent.

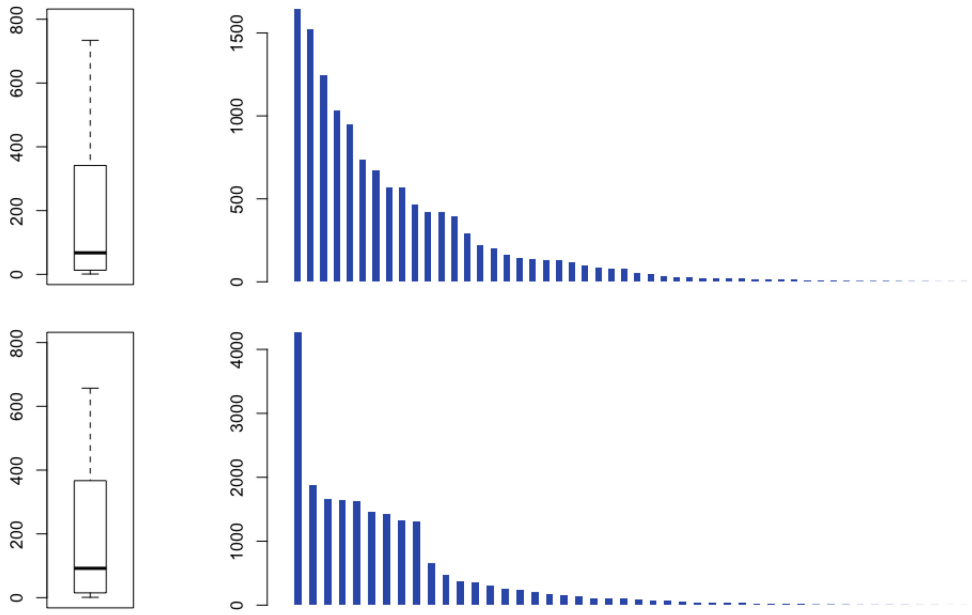

Fig. 10. Messages per user on Slack. *Top:* 2016 (M=248, SD=396). *Bottom:* 2015 (M=453, SD=808). The range of the box plots has been fixed, hiding outliers, to allow easier comparison of the 2 years.

the chart of people reading messages during the course in 2016. Although there is a gradual decline from the initial level of activity, and a small ramp close to the final project deadline, the peaks and troughs that were evident in the charts of messages posted are much less pronounced.

As well as message-writing behaviour not being uniform throughout the term, it is also not evenly distributed across the class. Figure 10 charts the messages posted on Slack per user in the 2015 and 2016 iterations of the course, anonymised and in descending order. In both cases, the message stream appears to be dominated by a number of vocal students, while there is a long tail of students who are present but post few messages. In both 2015 and 2016, 27 students posted at least 50 messages into Slack over the trimester. However, the very high variability in participation raises the concern that the project conversation may centre on a small portion of the class.

It is worth noting that Slack is not only used for interteam communication but also for intrateam communication, as teams with distance education students are usually geographically distant from each other, not just from the rest of the project. In 2016, 12 groups created channels for group communication. Accordingly, the long tail of students who are less active participants could indicate that some groups are less communicative, or could indicate that they chose to use a different

medium than Slack for their group communication, and were therefore less likely to be present at any given moment to engage in serendipitous public chat with members from other groups.

5.3 Issue Management, Wiki, and Continuous Integration

Although Slack is the highest volume communication mechanism in the class, it is essentially for ephemeral conversation. Both iterations of the course exceeded the 10,000 message limit that is Slack's default limit for how much of the conversation history is made accessible. Like the on-campus students at UQ, the UNE teams coordinated their work using a ticket manager (within GitHub in 2015 and GitLab in 2016), and documented their feature's design on the associated wiki. The move to GitLab in 2016 was logistical, allowing students to log in using their university accounts (via LDAP integration). This removed the need for staff to manage how students' university identities mapped to their GitHub identities.

In 2015, 50 issues were created by students in the issue manager over the trimester, attracting 555 comments. Of these, the 14 issues tagged *enhancement*, which represent the features the teams developed, received a greater number of comments (M = 20.6, SD = 13.6) as other teams coordinated with them. On the wiki, teams created 21 pages documenting their designs, with 40 students making 440 revisions across the trimester (M = 11.0, SD = 11.4). The continuous integration server attempted to build the students' code 288 times, with 214 successful builds.

The data from these systems suggests that they were used appropriately by students, and that there were reasonable levels of engagement and coordination between the teams. The differences in numbers between the 2014 UQ and 2015 UNE cohorts makes direct comparison on the same project difficult. When comparing with projects from the 2016 UQ tutorial groups, there appears to be similar levels of engagement. For example, one of the tutorial projects created 39 issues attracting 466 comments, and on the wiki 47 students made 570 revisions to 30 pages. Athough these on-campus numbers are somewhat larger, we do not find there to be qualitative differences between how the on-campus and distributed classes behave in regard to the issue manager or wiki.

6 CRITIQUE ANALYSIS

In 2016, we altered the critique process so that students would deliver their critiques via video, not just their presentations. We hypothesised that this would allow students to present their critiques more richly, for example, through the use of visual props and deixis. We also hope this will have a second benefit of promoting a sense of community within the course, as more students hear each other's voices. As part of our analysis of the distributed course at UNE, these critiques also became a subject of analysis.

6.1 Development of an Analysis Framework for the Critiques

In preparation for the analysis of the critiques, we undertook a literature review concerning peer and tutor feedback, focusing on higher education and upper secondary high school Information and Communication Technology (ICT) contexts.

A review by Winstone et al. [74] considered 30 years of literature to define and categorise *proactive recipience* processes and the characteristics of messages in feedback. Using a broad framework and coding system, the authors characterised assessment literacy to support the recipience process, a process highly relevant to our overarching focus to improve feedback dialogue and learner engagement. Emphasising a stepped assessment framework, Weltzer-Ward et al. [71] developed a theoretically based coding process to uncover the structure and quality of critical thinking in online discussions providing feedback. This coding process highlighted the value of developing a stepped solution for coding and analysis, seeking out prompts in feedback, and the importance of structure and quality not being mutually exclusive when assessing feedback.

Existing frameworks with potential relevance to our project included Tseng and Tsai [70], who analysed peer feedback using a framework by Chi [14] to categorise feedback into four categories: corrective, reinforcing, didactic, and suggestive. Gielen and De Wever [28] proposed a scoring rubric for analyzing peer feedback content based on the Feedback Quality Index [58] and later developed a coding scheme to structure the peer feedback process to include feedback category descriptions and examples [29]. Brown and Glover [10] developed a classification system to distinguish actionable feedback by identifying sub-categories to analyse feedback content and skills.

A subset of 48 videos were selected, comprising 12 videos documenting the team designs and 36 peer-critique videos authored by 12 students who each critiqued three team presentations. The teams and students were selected manually by three researchers, including the lecturer, in an attempt to maximise the variety of students that would be captured by the sample set. To develop our coding protocol, each of the aforementioned frameworks was applied to the video data to evaluate their usefulness in our context. In parallel, we also conducted a content and thematic analysis to identify alignment with the literature, and missing attributes that were not covered by the frameworks in the literature. Two researchers conducted the iterative video analysis by firstly selecting and coding the peer-critique videos, and then coding the team videos. The coding was done in this way to eliminate possible order bias that might come from watching comprehensive coverage of project elements in the team videos first. The analysis used a well-defined procedure adapted from the inductive approach to video research [21]. That is, we viewed the unedited video several times, with deeper analysis on subsequent viewings to uncover major themes. During the first level of analysis, we identified the broader characteristics. In the second level analysis, the team presentation and peer-critique videos were viewed to code the finer grained events.

The full research team convened to compare and contrast the literature framework analysis with the emergent themes, categories, and codes from the parallel content and thematic analysis. This indicated that three frameworks appeared particularly relevant to the current project: Brown and Glover [10] in relation to motivation feedback, Gielen and De Weaver [29] to help us define and refine categories to measure the quality of the peer feedback message, and Tseng and Tsai [70] to characterise the four types of feedback.

The thematic analysis led us to directly adopt the Tseng and Tsai [70] *Feedback Type*: corrective, reinforcing, didactic and suggestive. We redefined a subset of categories from Brown and Glover [10] relating to motivational praise, adding the categories of *evaluative* (judgement of performance) and *descriptive* (specific information describing particular qualities of the project). From Gielen and De Weaver [28], we adopted the categories of *suggestions* and *examples* from their scoring rubric.

The content analysis also identified several missing elements, leading to the development of four additional themes: *Presenter Team Focus*, *Reviewer Peer Critique Focus*, *Reviewer Presentation Mode*, and *Reviewer Quality of Peer Feedback*. All component categories are considered not to be mutually exclusive.

The final classification protocol for coding the data comprised 5 themes and 42 categories to characterise the main elements of the team and peer-critique videos. The final analysis iteration involved revisiting and re-coding the video data against these themes.

6.2 Critique Analysis Results

In this section, we present descriptive results from the coding of the videos against the classification protocol. For each theme, we present the categories that were identified within them, together with the results for each category. Some characteristics of the videos that do not directly relate to the categories but are practically helpful for teachers to know (such as the duration of the videos—whether the class adhered to the requested length) are also presented.

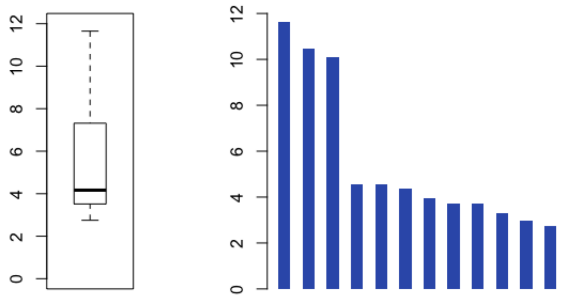

Fig. 11. Team video duration distribution (M = 5.51 minutes, SD = 3.22).

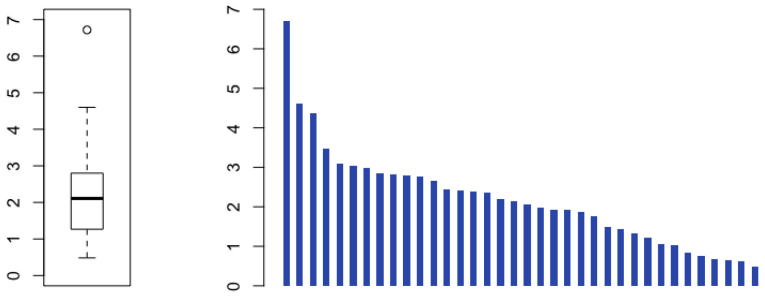

Fig. 12. Critique video duration distribution (M = 2.20 minutes, SD = 1.27).

6.2.1 Presenter Team Focus Theme. The team work-in-progress presentation videos ranged between 3 and 12 minutes in duration (5 minutes was requested in the assignment description) and the critique videos ranged between 1 and 4.5 minutes in duration (no length was formally specified). Figures 11 and 12 show a representation of the distribution.

The Presenter Team Focus theme comprised two categories of *code* representing coding language and *visual and mechanic* representing the aesthetics and functional aspects of the project. Ten out of twelve presenters primarily focused on *visual and mechanic* aspects of the project (80%) and *code* (20%), with the remaining two out of twelve presenters focusing on *visual and mechanic* (40%) and *code* (20%).

6.2.2 Reviewer Peer Critique Focus Theme. This theme comprised four categories: *visual* representing the aesthetics, *mechanic* representing the functional aspects, *code* representing coding language, and *structure* representing aspects of the actual video presentation. Table 1 summarises the results in this theme.

Across critique videos, regardless of the focus adopted by the team, the predominant peer-critique focus was on the mechanics of the project with a secondary focus on the visual aspects of the project. Although just over one-third of critiquers referred to coding, it was a low level focus.

6.2.3 Reviewer Presentation Mode Theme. Interestingly, all team videos displayed a computer screen slide-based presentation with spoken narration to document the team's design work. Whereas, the predominant Presentation Mode in the peer-critique reviewer videos displayed the *student's computer screen* (78%). This was followed by *face-to-camera* (15%) and *camera-to-paper* (7%) presentation. *Text-on-screen* featured as the major prop in all videos along with *supporting diagrams* (39%) and *images* (22%).

Table 1. Critique "Reviewer Focus" Category Distribution and Level of Focus Expressed within the Video

Categories	Overall percentage (n=36)	Level of focus expressed within category	
		% Low focus	% High focus
Visual	58% (21/36)	70% (15/21)	30% (6/21)
Mechanic	88% (32/36)	34% (11/32)	66% (21/32)
Coding	36% (13/36)	77% (10/13)	23% (3/13)
Structure	33% (12/36)	8% (1/12)	92% (11/12)

Note: Categories are not mutually inclusive.

Table 2. Feedback Type Definition Adapted from Tseng and Tsai [70]

Feedback Type	%
Reinforcing: Feedback on what is proper or correct. Positive feeling or recognition of the work expressed. Could include inexplicit encouragement without reason.	47% (17/36)
Suggestive: Indirect style advisory feedback if design is incomplete rather than incorrect. May alert author to problem without telling exactly what the problem is. Hints, pauses, rising intonation used to redirect student's thinking. This kind of feedback is also considered a kind of scaffolding.	33% (12/36)
Didactic: Provide lengthy explanations for errors or inadequate information. Lengthy explanation to direct students onto right track.	14% (5/36)
Corrective: If student's preliminary design or information is incorrect, then peer gives feedback to directly point out and/or corrected error.	6% (2/36)

6.2.4 Reviewer Critique Feedback Type Theme. Tsang and Tsai [70] provided a comprehensive suite of feedback types, which are reported in Table 2. Of note, the reviewers primarily (80%) adopted a social approach to deliver their message supported by encouragement and direction for improvement.

6.2.5 Reviewer Quality of Peer Feedback Theme. The Quality of Peer Feedback Theme comprised three dimensions: *Organisational Logic*, *Motivational Praise*, and *Suggestions*.

Organisational Logic: The structure of the critique was categorised to indicate clearness of the message. 72% (26/36) presented reasonable structure and clear logic with the remaining 28% (10/36) providing a video critique which was considered fragmented and lacking in continuity.

Motivational Praise: 95% provided *evaluative* motivational praise expressing judgement about achievement. 49% of reviewers went on to provide *descriptive* motivational praise and support within a context specific to the project. Further examination of this dimension revealed that the majority (74%) of reviewers provided substantial and specific critique.

Suggestions: Overall, 83% (30/36) of critique reviewers provided suggestions for future improvement to team projects. There were two categories differentiating the nature of the suggestions. *Concrete* suggestions provided immediate solutions for the more tangible aspects of the project, with 37% (11/30) only providing a purely *concrete* suggestion. *Abstract* suggestions provided a process or mechanism to review and improve the project, with 13% (4/30) only providing a purely *abstract* suggestion. A suggestion with an approximately

even balance between *concrete* and *abstract* suggestion was provided by 44% (13/30) of the reviewers and 6% (2/30) provided a predominantly *abstract* suggestion with some further links to *concrete* suggestions.

6.3 Critique Analysis Discussion

The results from our analysis of the videos appear to support our hypotheses for the use of asynchronous video as a medium for providing critique.

We had hypothesised that this would provide a richer medium, allowing students to include visual props and use techniques such as deixis. Although text on screen was the most common visual aid used in critique videos, a significant proportion also used supporting diagrams, images, or paper. While our secondary hypothesis, that using video as a critique medium would enhance the social aspects of the course is not directly measured in the coding scheme, this hypothesis is encouraged by our finding that the vast majority of students primarily adopted a social approach to deliver their message, using encouragement and suggestion, and by the fact that 95% of critique videos provided some form of motivational praise. For the UNE cohort, where students are distributed across Australia and overseas, the critique process is one of few occasions in which they would hear each other's voices. That it transpires that they hear significant amounts of constructive encouragement from those voices seems inherently positive.

The video critiques were found to be predominantly well-structured and clear, and to focus on higher level aspects of the teams' development work, such as the "mechanic" of their feature and its visual aspects, rather than delving into the lower level detail of the code or the structure of the video presentation. We find this encouraging, in terms of its match to the aims of studio critique in the course. The course is not simply a course in programming language skills, but in distributed software development, and studio critiques are intended as an avenue for reflective practice and encouraging higher level thinking. As the project is a shared code base in a collaborative project, the lower-level aspects around correction of the code can be addressed through the collaboration tools.

We also observe a possible logistical difference between using asynchronous video in distance education and the critique mechanism on campus at UQ. When teams are presenting synchronously in class, the length of their presentation can be controlled by the simple expedient that at the end of their allotted time they need to vacate the podium so that the next team can have its turn. With cloud video submission, the length cannot simply be controlled by scheduling, but must either be written as an extra check into the submission mechanism, or included in the mark scheme. In our results, three presentation videos exceeded 10 minutes duration, and three critique videos exceeded 4 minutes in length. Currently, we do not actively control video duration, but rely on viewers' ability to scrub ahead in the video or increase the playback speed.

7 RECOMMENDATIONS

In this section, we gather and summarise some recommendations that come from our experience running the course:

- Taking account in the marking of the quantity and frequency of commits and merges encourages the class to follow continuous integration practices. We find using a rubric assists with this, as the descriptors in the rubric clarify that it is a data-informed judgement on the contribution, quality, and consistency of the groups' work in the project, rather than a simple measure that could be gamed.
- The early weeks of the project, where students are familiarising themselves with their fellow groups, the project vision, the code, and the toolchain, can be particularly challenging to many students. Care needs to be taken in how students are introduced to the project and brought on board.

—There is a careful balance to be drawn in the size of the initial code base. While it is generally advantageous to keep this code base small so it is easy to learn, it needs to establish an initial vision for the project and make key architectural decisions that would otherwise delay the class from getting started.
—The steep learning curve of the early weeks in the project means that at the first checkpoint, some teams may have committed less code even though they are engaged with the project. To avoid unduly penalising these teams, recent editions of the UNE class conduct presentation and critique at the checkpoints, but only summatively assess each group's contribution across the project as a whole.
—There can also be a significant rush in the final week of the project. To mitigate the risk of a last minute incompatible change from another team fouling a team's demo with little time to recover, although we ask students to merge to master regularly, we recommend they record their demo from their (more controlled) group branch before they deal with the last few merges.
—We recommend asking students to score the critiques they receive against a set of categories. As well as prompting students—who may be caught up in their own plans for their development work—to view and consider the critiques they have received, it also gives the students performing the critique some simple guidance in what is expected. Generally, however, the scaffolding we give to students on how to present their work in progress, and how to structure and present a critique, is an area we are continuing to work to improve.
—We encourage students not to appear in the videos if there is something else that is relevant to show. While this might seem counter-intuitive in terms of building trust, and some students do record their critiques as a talking head, it encourages students to think about what is most helpful to show in the frame for what they want to say. It also normalises not appearing in the video, which is useful as occasionally an online student has a significant cultural or personal reason why they do not wish to appear on camera.
—Some student-submitted videos can significantly exceed the requested length. How this will be handled should be made clear to students, for example, whether the marker will stop watching altogether, or whether they reserve the right to start skimming and scrubbing forward in the video and will take the excess length as a factor in the marking.

8 CONCLUSIONS

The development of the course at both institutions over 6 years has produced a design that appears to be scalable from small to large classes, and from on-campus environments to geographically distributed distance education. In both institutions, the course is structured to operate with a limited amount of resources and staffing.

At the time of writing, the distributed course at UNE has completed three iterations. The changes that were needed to adapt it for a mixed on-campus and online cohort were minor. The critique process was altered to address the issue that students could not present their work to their peers synchronously, and a Slack chat group was added to enhance collaboration and coordination. However, around the same time, the UQ course also made two changes that coincidentally brought their on-campus course closer to the UNE course: they also introduced Slack as a chat mechanism, and by splitting the class into separate projects for each tutorial group they reduced the project size to a similar number of students as in the UNE version. A design committee, which was a UQ innovation in 2015, was also adopted into the UNE course in 2017.

The distributed and on-campus classes behave very similarly to each other in terms of when work occurs. Although the course represents only one-quarter of a student's full-time study load, in both versions of the course the commit distributions extend across more than a full-time working

week. Although the UNE class includes many students who study part-time and consume teaching materials asynchronously, they still display typical on-campus class-like behaviours, such as a large variability in the work performed from week to week and being strongly influenced by marking deadlines. The variability in when students work (both within the week and from week to week) is much larger than would be typical for a professionally conducted global software engineering project. Although the class requires the practices of distributed projects, in some respects it behaves distinctly as a class.

In our analysis of the critique videos, we find that students primarily adopt a social approach to giving feedback and advice, with encouragement and suggestion outweighing explanation and correction by a wide margin. The vast majority of critiques we examined included motivational praise and suggestions. Eighty-five percent of critiques displayed some form of visual prop on screen during the critique, taking advantage of the visual repertoire that video offers, with many including supporting diagrams and images. We also find that critiques focus primarily on the mechanics and visual aspects of their peers' work, with fewer critiques referring to detailed code, and even then only with low focus.

Overall, the supercollaborative studio design has transferred well to UNE's predominantly online teaching environment. Students are engaging appropriately with the collaboration systems and in critique of each other's work. As the number of students on the project is large, each student also has a sizeable source of peers to seek advice from. This takes advantage of the variability in skill level within the class, as less experienced students, even if their group-mates are also less experienced, can be assisted by more experienced peers from other groups.

Lessons from the online and on-campus courses continue to inform each other. Although we have not taken the approach of setting up joint projects between students across the two institutions, we continue to collaborate on improving the design of supercollaborative courses and we invite other interested institutions to contact us.

REFERENCES

[1] +Acumen. 2018. Introduction to Human-Centered Design. Retrieved January 17, 2018 from https://www.plusacumen.org/courses/introduction-human-centered-design.

[2] Rakesh Agrawal, Behzad Golshan, and Evimaria Terzi. 2014. Forming beneficial teams of students in massive online classes. In *Proceedings of the 1st ACM Conference on Learning Scale Conference*. ACM, 155–156.

[3] Eric Allen, Robert Cartwright, and Charles Reis. 2003. Production programming in the classroom. *SIGCSE Bulletin* 35, 1 (Jan. 2003), 89–93. DOI: https://doi.org/10.1145/792548.611940

[4] William Billingsley, Bing Ngu, Huy Phan, Nicolas Gromik, and Paul Kwan. 2016. Using a video-based critique process to support studio pedagogies in distance education—A tool and pilot study. In *Show Me The Learning. Proceedings ASCILITE 2016*, S. Barker, S. Dawson, A. Pardo, and C. Colvin (Eds.). 43–48.

[5] William Billingsley and Jim Steel. 2013. A comparison of two iterations of a software studio course based on continuous integration. In *Proceedings of the 18th ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE'13)*. ACM, New York, 213–218. DOI: https://doi.org/10.1145/2462476.2465592

[6] William Billingsley and Jim R. H. Steel. 2014. Towards a supercollaborative software engineering MOOC. *Companion Proceedings of the 36th International Conference on Software Engineering - ICSE Companion 2014*, 283–286. DOI: https://doi.org/10.1145/2591062.2591157

[7] Hilary Bradbury and Peter Reason. 2003. Action research – An opportunity for revitalizing research purpose and practices. *Qualitative Social Work* 2, 2 (2003), 155–175. DOI: https://doi.org/10.1177/1473325003002002003 arxiv:0710.4428v1

[8] Ricardo Britto, Claes Wohlin, and Emilia Mendes. 2016. An extended global software engineering taxonomy. *Journal of Software Engineering Research and Development* 4, 1 (Jun 2016), 3. DOI: https://doi.org/10.1186/s40411-016-0029-2

[9] Frederick P. Brooks. 1987. No silver bullet: Essence and accident of software engineering. *IEEE Software* 20 (1987), 12. DOI: https://doi.org/10.1109/MC.1987.1663532

[10] Evelyn Brown and Chris Glover. 2006. Evaluating written feedback. In *Innovative Assessment in Higher Education*, Cordelia Bryan and Karen Clegg (Eds.). Routledge, 81–91.

[11] Christopher N. Bull, Jon Whittle, and Leon Cruickshank. 2013. Studios in software engineering education: Towards an evaluable model. In *Proceedings of the International Conference on Software Engineering*. 1063–1072. DOI: https://doi.org/10.1109/ICSE.2013.6606656

[12] David Carrington and Soon-Kyeong Kim. 2003. Teaching software design with open source software. In *Proceedings of the 33rd Annual Frontiers in Education (FIE'03)*, Vol. 3. S1C–9–14. DOI: https://doi.org/10.1109/FIE.2003.1265910

[13] Dawn Chandler and Bill Torbert. 2003. Transforming inquiry and action: Interweaving 27 flavors of action research. *Action Research* 1, 2 (2003), 133–152. DOI: https://doi.org/10.1177/14767503030012002

[14] Michelene T. H. Chi. 1996. Constructing self-explanations and scaffolded explanations in tutoring. *Applied Cognitive Psychology* 10, July (1996), 1–15. DOI: https://doi.org/10.1002/(SICI)1099-0720(199611)10:7⟨33::AID-ACP436⟩3.3.CO;2-5

[15] Tony Clear, Sarah Beecham, John Barr, Mats Daniels, Roger McDermott, Michael Oudshoorn, Airina Savickaite, and John Noll. 2015. Challenges and recommendations for the design and conduct of global software engineering courses: A systematic review. *Proceedings of the 2015 ITiCSE on Working Group Reports*, 1–39. DOI: https://doi.org/10.1145/2858796.2858797

[16] Ivica Crnković, Ivana Bosnić, and Mario Žagar. 2012. Ten tips to succeed in global software engineering education. In *Proceedings of the 2012 34th International Conference on Software Engineering (ICSE'12)*. IEEE, 1225–1234.

[17] Philip Crowther. 2013. Understanding the signature pedagogy of the design studio and the opportunities for its technological enhancement. *Journal of Learning Design* 6, 3 (2013), 18–28. DOI: https://doi.org/10.5204/jld.v6i3.155

[18] Daniela Damian, Allyson Hadwin, and Ban Al-Ani. 2006. Instructional design and assessment strategies for teaching global software development: A framework. In *Proceedings of the 28th International Conference on Software Engineering*. ACM, 685–690. DOI: https://doi.org/10.1145/1134285.1134391

[19] Michael Docherty, Peter Sutton, Margot Brereton, and Simon Kaplan. 2001. An innovative design and studio-based CS degree. *Proceedings of the 32nd SIGCSE Technical Symposium on Computer Science Education (SIGCSE'01)* 33, 1 (2001), 233–237. DOI: https://doi.org/10.1145/364447.364591

[20] Thomas A. Dutton. 1987. Design and studio pedagogy. *Journal of Architectural Education* 41, 1 (1987), 16–25.

[21] Fredrick Erickson. 2006. Definition and analysis of data from videotape: Some research procedures and their rationales. *Handbook of Complementary Methods in Education Research* 3 (2006), 177–192. DOI: https://doi.org/10.1017/CBO9781107415324.004 arxiv:arXiv:1011.1669v3

[22] Jesús Favela and Feniosky Peña-Mora. 2001. An experience in collaborative software engineering education. *IEEE Software* 18 (2001), 47–53. DOI: https://doi.org/10.1109/52.914742

[23] Oliver Ferschke, Diyi Yang, Gaurav Tomar, and Carolyn Penstein Rosé. 2015. Positive impact of collaborative chat participation in an edX MOOC. In *International Conference on Artificial Intelligence in Education*. Springer, 115–124.

[24] Luiz Leandro Fortaleza, Tayana Conte, Sabrina Marczak, and Rafael Prikladnicki. 2012. Towards a GSE international teaching network: Mapping global software engineering courses. *Proceedings of the 2012 2nd International Workshop on Collaborative Teaching of Globally Distributed Software Development (CTGDSD'12)*, 1–5. DOI: https://doi.org/10.1109/CTGDSD.2012.6226944

[25] Armando Fox and David Patterson. 2012. Crossing the software education chasm. *Communications of the ACM* 55, 5 (2012), 44. DOI: https://doi.org/10.1145/2160718.2160732

[26] Armando Fox, David A. Patterson, Richard Ilson, Samuel Joseph, Kristen Walcott-Justice, and Rose Williams. 2014. *Software Engineering Curriculum Technology Transfer: Lessons Learned from MOOCs and SPOCs*. Technical Report EECS-2014-17. Electrical Engineering and Computer Sciences, University of California at Berkeley.

[27] Paul Gestwicki and Brian McNely. 2016. Interdisciplinary projects in the academic studio. *ACM Transactions on Computing Education* 16, 2 (2016), 1–24. DOI: https://doi.org/10.1145/2732157

[28] Mario Gielen and Bram De Wever. 2012. Peer assessment in a wiki: Product improvement, students' learning and perception regarding peer feedback. *Procedia - Social and Behavioral Sciences* 69, Iceepsy (2012), 585–594. DOI: https://doi.org/10.1016/j.sbspro.2012.11.450

[29] Mario Gielen and Bram De Wever. 2015. Structuring peer assessment: Comparing the impact of the degree of structure on peer feedback content. *Computers in Human Behavior* 52 (2015), 315–325. DOI: https://doi.org/10.1016/j.chb.2015.06.019

[30] Olly Gotel, Vidya Kulkarni, Moniphal Say, Christelle Scharff, and Thanwadee Sunetnanta. 2012. Quality indicators on global software development projects: Does 'getting to know you' really matter? In *Journal of Software: Evolution and Process* 24 (2012), 169–184. DOI: https://doi.org/10.1002/smr.474

[31] Orit Hazzan. 2002. The reflective practitioner perspective in software engineering education. *Journal of Systems and Software* 63, 3 (2002), 161–171. DOI: https://doi.org/10.1016/S0164-1212(02)00012-2

[32] Dean Hendrix, Lakshman Myneni, Hari Narayanan, and Margaret Ross. 2010. Implementing studio-based learning in CS2. *Proceedings of the 41st ACM Technical Symposium on Computer Science Education*, 505–509. DOI: https://doi.org/10.1145/1734263.1734433

[33] Christopher D. Hundhausen, N. Hari Narayanan, and Martha E. Crosby. 2008. Exploring studio-based instructional models for computing education. *ACM SIGCSE Bulletin* 40, 1 (2008), 392. DOI: https://doi.org/10.1145/1352322.1352271

[34] Panayiotis Koutsabasis and Spyros Vosinakis. 2012. Rethinking HCI education for design: Problem-based learning and virtual worlds at an HCI design studio. *International Journal of Human-Computer Interaction* 28, 8 (2012), 485–499. DOI: https://doi.org/10.1080/10447318.2012.687664 arxiv:arXiv:gr-qc/9809069v1

[35] Thomas Kvan. 2001. The pedagogy of virtual design studios. *Automation in Construction* 10, 3 (2001), 345–353. DOI: https://doi.org/10.1016/S0926-5805(00)00051-0 {CAADRIA}.

[36] Patricia Lago, Henry Muccini, and Muhammad Ali Babar. 2008. Developing a course on designing software in globally distributed teams. In *Proceedings of the 2008 3rd IEEE International Conference Global Software Engineering (ICGSE'08)*. 249–253. DOI: https://doi.org/10.1109/ICGSE.2008.26

[37] Jerry Laiserin. 2002. From atelier to e-telier: Virtual design studios. *Architectural Record* 190, 1 (2002), 141.

[38] Phillip A. Laplante. 2006. An agile, graduate, software studio course. *IEEE Transactions on Education* 49, 4 (2006), 417–419. DOI: https://doi.org/10.1109/TE.2006.879790

[39] Jouni Lappalainen, Nirnaya Tripathi, and Jouni Similä. 2016. Teaching a global software development course: Student experiences using onsite exercise simulation. In *Proceedings of the 38th International Conference on Software Engineering Companion (ICSE'16)*. ACM, New York, 440–450. DOI: https://doi.org/10.1145/2889160.2889198

[40] Alan Levy. 1980. Total studio. *Journal of Architectural Education* 34, 2 (1980), 29–32. DOI: https://doi.org/10.1080/10464883.1980.10758247

[41] Kurt Lewin. 1946. Action research and minority problems. *Journal of Social Issues* 2, 4 (1946), 34–46. DOI: https://doi.org/10.1111/j.1540-4560.1946.tb02295.x

[42] Nan Li, Himanshu Verma, Afroditi Skevi, Guillaume Zufferey, Jan Blom, and Pierre Dillenbourg. 2014. Watching MOOCs together: Investigating co-located MOOC study groups. *Distance Education* 35, 2 (2014), 217–233.

[43] Yang Li, Stephan Krusche, Christian Lescher, and Bernd Bruegge. 2016. Teaching global software engineering by simulating a global project in the classroom. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education (SIGCSE'16)*, 187–192. DOI: https://doi.org/10.1145/2839509.2844618

[44] Peter Lloyd. 2013. Embedded creativity: Teaching design thinking via distance education. *International Journal of Technology and Design Education* 23, 3 (2013), 749–765. DOI: https://doi.org/10.1007/s10798-012-9214-8

[45] Judith Grant Long. 2012. State of the studio: Revisiting the potential of studio pedagogy in U.S.-based planning programs. *Journal of Planning Education and Research* 32, 4 (2012), 431–448. DOI: https://doi.org/10.1177/0739456X12457685

[46] Roger McDermott, Mats Daniels, Åsa Cajander, Mats Cullhed, Tony Clear, and Cary Laxer. 2012. Student reflections on collaborative technology in a globally distributed student project. In *2012 Frontiers in Education Conference Proceedings*. 1–6. DOI: https://doi.org/10.1109/FIE.2012.6462410

[47] Bertrand Meyer and Marco Piccioni. 2008. The allure and risks of a deployable software engineering project: Experiences with both local and distributed development. In *Proceedings of the IEEE 21st Conference on Software Engineering Education and Training (CSEET'08)*. IEEE, 3–16. DOI: https://doi.org/10.1109/CSEET.2008.41

[48] Miguel J. Monasor, Aurora Vizcaíno, Mario Piattini, and Ismael Caballero. 2010. Preparing students and engineers for global software development: A systematic review. In *Proceedings of the 2010 5th IEEE International Conference on Global Software Engineering (ICGSE'10)*. DOI: https://doi.org/10.1109/ICGSE.2010.28

[49] Miguel Jiménez Monasor, Aurora Vizcaíno, Mario Piattini, John Noll, and Sarah Beecham. 2014. Assessment process for a simulation-based training environment in global software development. In *Proceedings of the 2014 Conference on Innovation & Technology in Computer Science Education (ITiCSE'14)*. ACM, New York, 231–236. DOI: https://doi.org/10.1145/2591708.2591747

[50] Christian Murphy, Dan Phung, and Gail Kaiser. 2008. A distance learning approach to teaching extreme programming. *SIGCSE Bulletin* 40, 3 (June 2008), 199–203. DOI: https://doi.org/10.1145/1597849.1384325

[51] John Noll, Sarah Beecham, and Ita Richardson. 2011. Global software development and collaboration: Barriers and solutions. *ACM Inroads* 1, 3 (Sept. 2011), 66–78. DOI: https://doi.org/10.1145/1835428.1835445

[52] Tom Nurkkala and Stefan Brandle. 2011. Software studio: Teaching professional software engineering. *Proceedings of the 42nd ACM Technical Symposium on Computer Science Education (SIGCSE'11)*, 153–158. DOI: https://doi.org/10.1145/1953163.1953209

[53] Daniel F. O. Onah, Jane Sinclair, and Russell Boyatt. 2014. Dropout rates of massive open online courses: Behavioural patterns. *EDULEARN14 Proceedings*, 5825–5834.

[54] Maria Paasivaara, Kelly Blincoe, Casper Lassenius, Daniela Damian, Jyoti Sheoran, Francis Harrison, Prashant Chhabra, Aminah Yussuf, and Veikko Isotalo. 2015. Learning global agile software engineering using same-site and cross-site teams. In *Proceedings of the International Conference on Software Engineering*, Vol. 2. 285–294. DOI: https://doi.org/10.1109/ICSE.2015.157

[55] Şule Taşlı Pektaş. 2015. The virtual design studio on the cloud: A blended and distributed approach for technology-mediated design education. *Architectural Science Review* 58, 3 (2015), 255–265. DOI: https://doi.org/10.1080/00038628.2015.1034085

[56] Feniosky Peña-Mora, Rhonda Struminger, Jesús Favela, and Robin Losey. 2000. Supporting a project-based, collaborative, distance learning lab. In *Proceedings of the 8th International Conference on Computing in Civil and Building Engineering*, Vol. 279, 170–176. DOI: https://doi.org/10.1061/40513(279)22

[57] Javier Portillo-Rodríguez, Aurora Vizcaíno, Mario Piattini, and Sarah Beecham. 2012. Tools used in global software engineering: A systematic mapping review. *Information and Software Technology* 54, 7 (2012), 663–685. DOI: https://doi.org/10.1016/j.infsof.2012.02.006

[58] Frans J. Prins, Dominique M. A. Sluijsmans, and Paul A. Kirschner. 2006. Feedback for general practitioners in training: Quality, styles, and preferences. *Advances in Health Sciences Education* 11, 3 (2006), 289–303. DOI: https://doi.org/10.1007/s10459-005-3250-z

[59] Ita Richardson, Allen E. Milewski, Neel Mullick, and Patrick Keil. 2006. Distributed development: An education perspective on the global studio project. In *Proceedings of the 28th International Conference on Software Engineering (ICSE'06)*. ACM, New York, 679–684. DOI: https://doi.org/10.1145/1134285.1134390

[60] Miguel Romero, Aurora Vizcaíno, and Mario Piattini. 2008. Using virtual agents for the teaching of requirements elicitation in GSD. In *Lecture Notes in Computer Science (including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, Vol. 5208 LNAI. 539–540. DOI: https://doi.org/10.1007/978-3-540-85483-8_78

[61] Farnaz Ronaghi, Amin Saberi, and Anne Trumbore. 2015. NovoEd, a social learning environment. In *Massive Open Online Courses: The MOOC Revolution*, Paul Kim (Ed.). Routledge, New York, Chapter 7, 96–105.

[62] David Root, Mel Rosso-Llopart, and Gil Taran. 2008. Exporting studio: Critical issues to successfully adopt the software studio concept. In *Software Engineering Education Conference, Proceedings*. 41–50. DOI: https://doi.org/10.1109/CSEET.2008.21

[63] Marc Aurel Schnabel and Jeremy J. Ham. 2012. Virtual design studio within a social network. *Electronic Journal of Information Technology in Construction* 17 (2012), 397–415.

[64] Donald A. Schön. 1984. The architectural studio as an exemplar of education for reflection-in-action. *Source Journal of Architectural Education* 38, 1 (1984), 2–9. DOI: https://doi.org/10.1080/10464883.1984.10758345

[65] Donald A. Schön. 1987. *Educating the Reflective Practitioner: Toward a New Design for Teaching and Learning in the Professions*. 355 pages.

[66] Darja Šmite, Claes Wohlin, Zane Galvina, and Rafael Prikladnicki. 2014. An empirically based terminology and taxonomy for global software engineering. *Empirical Software Engineering* 19 (2014). DOI: https://doi.org/10.1007/s10664-012-9217-9

[67] Jörn Guy Süß and William Billingsley. 2012. Using continuous integration of code and content to teach software engineering with limited resources. In *Proceedings of the International Conference on Software Engineering*. 1175–1184. DOI: https://doi.org/10.1109/ICSE.2012.6227025

[68] James E. Tomayko. 1991. Tomayko: Teaching software development in a studio environment. In *Proceedings of the 22nd SIGCSE Technical Symposium on Computer Science Education (SIGCSE'91)*, Vol. 23. 300–303. DOI: https://doi.org/10.1145/107005.107070

[69] James E. Tomayko. 1996. Carnegie Mellon's software development studio: A five year retrospective. In *Proceedings of the 9th Conference on Software Engineering Education*. 119–129. DOI: https://doi.org/10.1109/CSEE.1996.491367

[70] Sheng-Chau Tseng and Chin-Chung Tsai. 2007. On-line peer assessment and the role of the peer feedback: A study of high school computer course. *Computers & Education* 49 (2007), 1161–1174. DOI: https://doi.org/10.1016/j.compedu.2006.01.007

[71] Lisa Weltzer-Ward, Beate Baltes, and Laura Knight Lynn. 2009. Assessing quality of critical thought in online discussion. *Campus-Wide Information Systems* 26 (2009), 168–177. DOI: https://doi.org/10.1108/10650740910967357

[72] Miaomiao Wen, Diyi Yang, and Carolyn Penstein Rosé. 2015. Virtual teams in massive open online courses. *Lecture Notes in Computer Science (including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 9112 (2015), 820–824. DOI: https://doi.org/10.1007/978-3-319-19773-9_124

[73] Betsy Anne Williams. 2015. Peers in MOOCs. In *Proceedings of the 2nd 2015 ACM Conference on Learning @ Scale (L@S'15)*. 287–292. DOI: https://doi.org/10.1145/2724660.2728677

[74] Naomi E. Winstone, Robert A. Nash, Michael Parker, and James Rowntree. 2016. Supporting learners' agentic engagement with feedback: A systematic review and a taxonomy of recipience processes. *Educational Psychologist* (2016), 1–21. DOI: https://doi.org/10.1080/00461520.2016.1207538

Received April 2017; revised January 2018; accepted March 2018