2015 IEEE/ACM 37th IEEE International Conference on Software Engineering

Software Design Studio: A Practical Example

Jaejoon Lee, Gerald Kotonya, Jon Whittle, Christopher Bull
School of Computing and Communications
Lancaster University
Lancaster, UK
{j.lee3, g.kotonya, j.n.whittle, c.bull}@lancaster.ac.uk

***Abstract*—We have been generally successful for transferring software engineering knowledge to industry through various forms of education. However, many challenges in software engineering training remain. A key amongst these is how best to energise software engineering education with real-world software engineering practices. This paper describes our experience of delivering a radically different approach based on the notion of a Software Design Studio. The Software Design Studio is both a lab for students engaged in conceiving, designing and developing software products as well as an approach for teaching software engineering in the lab which emphasizes practical hands-on work and experimentation. The feedback on the Software Design Studio – from both staff and students – has been outstanding. Although the programme is designed as a small, elite programme there is interest to see if the teaching methods can be transferred across to the much larger undergraduate programme in Computer Science. In this paper, we provide a detailed description of how our studio works in practice so that others, thinking of taking a studio or studio-inspired approach, can use in designing their own courses.**

***Index Terms*—Software engineering education, software design studio, reflective teaching approach.**

I. Introduction

Software engineering has evolved greatly from its humble beginnings in the late 1950s when it was largely defined by coding. Today software engineering is concerned with how best to create software in ways that promote sustainable development and maximize quality. This evolution has been influenced by 40 years of advances in software engineering including; abstraction, modularity, generality, anticipation of changes, separation of concerns and several software development paradigms. There is general consensus amongst researchers and practitioners alike that these advances have greatly improved the quality and delivery of the software produced today [1]. This means that we have been generally successful for transferring software engineering knowledge to industry through various forms of education [2] [3]. However, many challenges in software engineering training remain. A key amongst these is how best to energise software engineering education with real-world software engineering practices.

This paper describes our experience of delivering a radically different approach based on the notion of a *Software Design Studio*. The *Software Design Studio* is both a lab for students engaged in conceiving, designing and developing software products as well as an approach for teaching software engineering in the lab which emphasizes practical hands-on work and experimentation. Contact time with teaching staff is in the form of lab time that emphasises problem solving, practical skills grounded in theory, peer critique, industry involvement and mentoring.

Our other papers have focused on results from the 1.5yr study, focusing on individual aspects of the studio experience, such as how it encourages reflective practice [5], how students learn design and modeling in a studio [6], and experiences from other software engineering studios [7]. However, none of our other papers have given a detailed description of how our studio works in practice; we do so in this paper. The contribution of the paper, therefore, is a concrete description of a studio, which others, thinking of taking a studio or studio-inspired approach, can use in designing their own courses.

The rest of this paper is organized as follows; section II describes the motivation for the *Software Design Studio*. Section III describes the structure of the *Software Design Studio*. Section IV and V provide the student feedback and lessons learned. We discuss related work in Section VI and Section VII provides some concluding thoughts.

II. Motivation

Studio education originates from disciplines such as architecture, design, and art, with a relatively recent uptake from software-based disciplines. In those disciplines students work on projects that are often visual, collaboration is common, and discussions and critique between students and staff is a frequent occurrence. Because of this, studios are often described as a second "home" for students, as they tend to spend a lot of time in these environments. They are also used to emphasize learning-by-doing, which is in contrast to a traditional lecture-based approach. In [6], we have explored the complex understandings of studio education, through interviews with teaching staff and practitioners from the disciplines that studios originated from. This work culminated in the creation of the 'studio framework', which describes what the interview participants considered core facets of studio education, providing a clearer understanding of the approach.

Correctly implemented studios offer numerous benefits to students due to its practical nature and emphasis on the importance of problem solving. Software practitioners are expected to find solutions to large and complex problems whilst often working within teams; a studio helps prepare students in this regard as it offers an opportunity for them to learn and become comfortable with numerous reflective practice techniques [5],

978-1-4799-1934-5/15 $31.00 © 2015 IEEE
DOI 10.1109/ICSE.2015.171

ICSE 2015, Florence, Italy
Joint SE Education and Training

such as coaching and critiquing. Students that present their work often and experience frequent formal and informal critique from teaching staff and peers learn to better apply a critical eye over their future work. This also has an implicit effect on students' soft skills, such as collaboration and communication skills, which ultimately make for a better software engineer.

A software design studio provides the opportunity to practice software engineering principles and concepts in the context of large complex projects, which inherently require them. Practicing these principles in limited projects and sterile environments is not realistic; therefore their importance is learnt out of context. It is therefore potentially easy for students to dismiss the relevance of software engineering unless they are experienced in the context of complex software design problems. One way we have specifically experienced this is by students asking why they simply cannot 'just do it.'

Despite the numerous benefits that studio education can provide, professional competency for example, an oft discussed reason for pursuing studio education is to make teaching and learning software engineering more interesting. An example is the constant communication amongst peers, in relation to the spaces description as a second 'home,' providing students with opportunities to be more critical of their work but also making the experience more social. Another is that students get to work on large projects and complex designs, providing students with the opportunity to experience elements of exploration. In this regard, teaching staff has the opportunity to experience students' creativity, especially when they are given the ability to define and pursue their own projects.

III. SOFTWARE DESIGN STUDIO

The undergraduate software engineering degree program at Lancaster is structured over three years. The Software Design Studio builds on the knowledge and experience gained in Year 1 and Year 2 of study that introduce students to programming and testing (SCC.110: Software Development), and software lifecycle models, requirements engineering, software architecture and design patterns (SCC.204 Software Design). In Year 3, the focus is on the Software Design Studio.

The Software Design Studio begins in Year 2 and is taught in three modules: SCC.230: Core Studio module for second year students, and SCC.330: Networked Studio and SCC.331: Live Studio modules for third (final) year students. All modules involve a group project with teams of 4-6 students.

In SCC.230 (Core Studio), students choose their own project topics and 3.5 hour workshops run every two weeks. In the first workshop, students come up with their own ideas for their product. This is done as a facilitated creativity session. Then the students work on their projects in the workshops, building and testing parts of their product relevant to that theme. These workshops focus on either technical skills – software requirements, design and implementation – or transferable skills. For example, one session is 'acting skills for presenters' in which an actor teaches basic acting skills that students can use to improve their presentation skills.

In SCC.330 and SCC.331 (Networked and Live Studios), students work on a much larger project, closer in scale to industrial problems. This last academic year, for example, students used a variety of sensors and programmable devices to convert their lab into a 'smart lab'. A key novelty of SCC.330 and SCC.331 is the use of agile development, a technique well known in industry, to drive the projects. Agile methods involve weekly deliverables of parts of the product, built-in reflection on what could be done better the next week, and tracking personal productivity through a series of 'burn down charts.' In short, in the studio modules, a variety of novel methods are applied that take precedence over traditional lectures – agile methods, flipped classroom, acting skills, creativity methods, and video – that make the modules engaging, interesting and a valuable learning experience for students. In the following, we explain them in terms of project, schedule and deliverables, and assessment.

A. SCC.230:Core Studio

- Projects: The students came up with three project ideas: a mobile lecture feedback system, a web-based business start-up ideas and angel investors matching system and a smart bus rider support system. The general guidelines we provide for the students when they decide their projects during the first workshop session are: 1) the system should be of sufficient size and complexity for a 150 hour, 15 credit module, 2) the assessed grade will take into account the size and complexity of the system design and implementation, 3) the emphasis is on ideation, planning, design, implementation, testing, documentation and presentation, 4) the workshop sessions will focus on introducing broader software engineering concepts and issues and, therefore, the students are required to expand their programming skills as part of their self-study (although, of course, the mentors will provide assistance as and when appropriate).

- Schedule and Deliverables:

Week	Workshop Theme	Theme Description
1	Introduction to SCC.230 & Project Selection	• Brief introduction to SCC230 teaching methods/assessment • Group creativity exercise for group formation and selection of group projects • Outcome: students have chosen their project teams and have initial concept for their team project • Homework: learn about iterative/incremental development processes & requirements engineering
2	Planning a Project	• Iterative/Incremental development processes: students work up their plans during the session • Project Management Processes (e.g., PSP) & Expectations on Documenting Group Projects • Requirements Specification:

		students work up their plans during the session • Outcome: projects have a project plan and initial set of requirements • Homework: Complete the set of requirements; read up on software architecture
3	Software Architecture	• Students work on their software architecture during the session • Outcome: initial software architecture • Homework: read up for next session
4	First increment: design choices	• Hands-on design session for first increment
5	First increment: Testing	• Students should by now have an implementation of the first increment and conducted some basic testing by themselves. This session focuses on more structured testing methods; with hands-on application of these methods. Should include peer testing (i.e., groups take on role of testers for other groups)
6	Second increment: design and testing	• Further hands-on design and testing • Design patterns
7	Third increment: design and testing	• Further hands-on design and testing • Design patterns
8	Requirements Changes	• Introduce a change to the requirements • Session explores how best to handle requirements changes and how to update design/implementation • Should include some consideration of quality attributes
9	Improving the Software	• Session looking back at what the groups have produced so far and actively looking for ways to improve the software quality
10	Presentation skills	• Advice on how to present their projects in the final presentations

- Assessment: Marks are awarded for the different parts of the project. These consist of both a group mark and an individual mark as follows: 1) Assessment of the project as a whole counts for 40% of the total marks. This is assessed on a group report, a working demo and/or an oral presentation, 2) Each student produces an individual portfolio describing his/her personal contribution to the project and this is 40% of the total marks. The group report summarizes the work, whereas the individual report goes into technical detail, 3) Assessment of an individual student based on the group project mentor's report on the student counts for 20% of the overall marks. The mentors will base their report on the student's participation and contributions to the group project and workshop activities.

B. SCC.330: Networked Studio

-Project: This module build on the knowledge and skills gained in SCC.230 to introduce the students to the development of large software systems with more realistic requirements. It focuses on the integration and networking of software modules to create larger systems. In particular, the students learn software engineering techniques relevant to medium scale networked projects such as models of distributed architecture, large-scale integration testing, distributed team development, and techniques for large scale software quality.

To this end, we select a smart lab project – the aim of the Smart Lab System (SLS) is to turn the Software Design Studio lab into a smart interactive lab. The development process is spread over 10 weeks and follows an agile process with a working system release every week from Week 2. In the first phase of the project the students develop a number of basic services to monitor and collect information on climatic changes, interactions and movements in the lab. In the second phase of the project they use the services to compose smart lab applications, which they will in integrate into the smart lab.

- Schedule and Deliverables:

Week	Deliverables
3	- The service shall provide, on-demand, the temperature and light values in a specific zone in the lab. The lab shall be divided into 3 zones.
	- The service shall provide, on demand, an hourly history of temperature and light changes in the lab for up to one week.
4	- The service shall provide, on demand, the time that specific objects in the lab were last accessed or interacted with
	- The service shall provide, on demand, the history of when the specific objects in the lab were accessed or interacted with, for up to one week
5	- The service shall provide, on demand, the location of a lab user
6	- The service shall provide, on demand, an hourly movement history (i.e. between different the zones) of a lab user, for up to one week.
	- The service shall provide on demand, the ID of a pressed button press together with time (i.e. Sw1 or Sw2 on Sun SPOT)
7	- It shall be possible to interact with the SLS via the *Touch Table* and desktop computer

	- The system shall provide a real-time visualisation of the temperature and light changes in the four lab zones.
8	- The system shall provide a means to display a comparative history graph showing changes in temperature over time
	- The system shall provide a means to display a comparative history graph showing changes in light over time
	- The system shall provide a visualisation showing objects in the four zones and indicating when they were last accessed/interacted with
9	- The system shall provide a visualisation of lab user location
	- The system shall provide a visualisation of lab user movement (i.e. history of lab user movement)
	- The system shall provide periodic Twitter commentary of the happenings in the lab; including climactic, environment, location and activity information.
10	- The system shall use the temperature service to control heating devices and fans in the lab The device control should reflect temperature changes in the 3 zones.
	- The system shall use the light monitoring service to control lighting in the lab. The device control should reflect light changes in the 3 zones.
	- Smart cup: This cup can monitor the daily intake of fluids. This can be implemented by attaching a Sun SPOT to a cup and the smart cup monitors the fluid level of the cup by measuring the tilted angle of the cup. This smart cup reports the daily intake of fluids when requested. Further, the tilting behaviour of the cup can be visualized in real-time.

The first two weeks are dedicated to get to know the Sun SPOT devices [8], which are used as sensors/actuators of SLS, and the Scrum agile process [9].

- Assessment: The assessment begins from week 3 and the marks consist of both a group mark (30%) and an individual mark (70%). We have defined a weekly sprint of the Scrum framework and the assessment (group as well as individual) also happens weekly. Each group is expected to demonstrate a running system and individual contribution to the group work is assessed after the demo. Then we have a one-to-one feedback session for each student. During the individual feedback session, we provide detailed description on what went well and what needs to be improved. For example, we discussed the each section of reflective reports, which were submitted by the students, and gave written and verbal comments on the students' activities. The reflective report consists of four sections including 1) Detailed Description of Deliverables, 2) Delivered Items, 3) Not Delivered Items, and 4) Impediments and Plans to Address Them. This personalized and prompt feedback is one of the major factors for the success of this module.

C. SCC.331: Live Studio

-Project: SCC331 extends the smart lab project from SCC.330 to develop a generic toolkit for smart living and smart working (SET: Smart Environment Toolkit). Like SCC.330, SCC.331 is a 30-credit module with a focus on engineering a non-trivial software system. We have adopted the Scrum framework as the development process with a strong emphasis on disciplined development.

To underscore the importance of engineering principle and real-world practice, we have a strong focus on the following:

1) Business context through interaction with potential users and organisations that are involved in the development of similar products: The students can learn to adapt their knowledge and skills to shifting business goals and needs (e.g. functionality, cost, time to market and quality);

2) Software engineering principles: A key aim of software engineering is to use sound engineering principles in order to obtain cost-effective software systems that are reliable and work efficiently. The SET project is underpinned by software engineering principles and the students can learn essential component in a software engineer's toolset (e.g. good coding practice, testing, modularity, design patterns, anticipation of change, generality and ability to deliver subsets of the system early to obtain user feedback); and,

3) Professional competency: Whereas the training of software engineers traditionally emphasizes technical skills, it is also important that software engineers have broad professional competencies. These include the ability to communicate complex concepts effectively, the ability to work well with others, and the ability to adapt knowledge and skills to address emerging problems.

- Schedule and Deliverables:

Week	Workshop Theme	Theme Description
1	Introduction to SCC331	• Brief introduction to SCC331 • Objectives and Project • Badge system • Assessment • Teams start designing the SET architecture
2	Industry Workshop	• A guest speaker from industry gives a talk on their business followed by a demo of one of their products • Each team demonstrates their system to the guest. • Comments and feedback from the guest.
3	Demo of system release #1	• Demo of system release # 1: The SET shall provide a setup mechanism that automatically recognises new [active] Sun SPOT sensors in

		the user environment. The mechanism shall recognise both the sensor address and the type of data it collects (e.g. light, temp, tilt, infrared etc.). • Retrospect
4	Demo of system improvement #1	• Demo of improvement for release # 1 (individual) • Retrospect
5	Demo of system release #2	• Demo of system release # 2: The SET shall provide the user with a mechanism for expressing the trigger conditions for high-level services (i.e. security, safety, comfort, health, and reminder services). A trigger shall use logical and relational operators to create a comparison. A trigger specification shall be able to use all active sensor sources. The SET shall provide a mechanism for writing workflow descriptions to a database server, and for accessing and updating them. The SET App shall provide graphical functionality for displaying the status of different SET devices in the user environment. • Retrospect
6	Demo of system improvement #2	• Demo of improvement for release # 2 (individual) • Retrospect
7	Demo of system release #3	• Demo of SET release # 3: The SET App shall provide a graphical user interface to allow direct control of the SET actuators in the user environment. The SET App shall provide functionality for visualising the data trends in the user environment. The SET App shall provide functionality for supporting reminder and notification services. • Retrospect
8	Demo of SET System improvement #3	• Demo of improvement for release # 3 (individual) • Retrospect
9	Demo of system release #4	• Demo of system release # 4: The SET shall provide functionality for monitoring and executing the service trigger conditions. The SET shall provide functionality for managing client registration, trigger definitions and sensor data. • Retrospect • Requirement for the project video
10	Demo of system project video	• Demo of system project video (group) • Demo of improvement for release # 4 (individual) • Retrospect

- Assessment: The previous SCC.330 (Networked Studio) has been extremely well received by the students and it appears that they are both engaged with the course and achieving a high level of learning. However, even though they have been taught software engineering (SE) principles (both in SCC.230 and SCC.330), because these have been taught through practice, the students are *not* always able to explicitly articulate their learning goals in terms of SE principles. Therefore, we more focus on individual attainment of software engineering principles as well as system implementation techniques in SCC.331. The assessment is designed to support this focus and we also have introduced software engineering badges. This badge-based assessment is intended to make the learning of SE principles more explicit and would force students to articulate the SE principles they have learned. As we assess them explicitly on these principles, it would also give students explicit SE principles that they can list on their CVs, thus increasing employability.

Each 'badge' corresponds to a software engineering principle for which students should demonstrate competency. The students are expected to collect a minimum of four badges and the assessment is based on these badges. The first and second badges are compulsory and consist of Personal Software Process (PSP) level 0.0 and 0.1, respectively. The third and fourth badges can be chosen from the following: 1) refactoring, 2) test-driven development, 3) reverse engineering software architecture, 4) design patterns, 5) automated testing, 6) usability testing and 7) code quality assessment by metrics. We came up with these badge items with consideration of [10] and our experience with industrial partners.

The marks for the module are structured as follows:

• (20%) Assessment of the group deliverables as a whole will count for 20% of the total marks. This is assessed on each group's working demo (source code) and oral presentation;
• (40%) Each student produces an individual portfolio describing his/her personal contribution to the project. We have demo sessions for individual students after the group demos/presentations. This individual session should include technical details about the contribution; and,
• (40%) Each student produces an individual portfolio describing his/her personal contribution to the improvement of the project through the software engineering badge scheme. PSP level 0.0 and level 0.1 are compulsory for all students. Students can select two optional badges from the seven software engineering badges.

D. Workshop Activities and Delivered Products

Workshops focus on creativity, technical skills, software requirements, design, implementation, and transferable skills.

Figure 1 captures some of the outputs from these workshops (. For example, Fig. 1a) is from the project idea session: all students propose their ideas and we classify them into a number of themes (e.g., health and fitness, student/staff relation, etc.). Then the students vote to pick the most popular ideas. For the group forming workshop (Fig. 1b), each student presents his/her experience from Year 1 in terms of i) what they liked, ii) what they liked least, iii) their strengths and iv) what would like to learn through the module. Then they visit each other's profile and find appropriate group members who can complement their own skill sets. One of the project ideas was a mobile lecture feedback system and Fig. 1c) is the final product from this project.

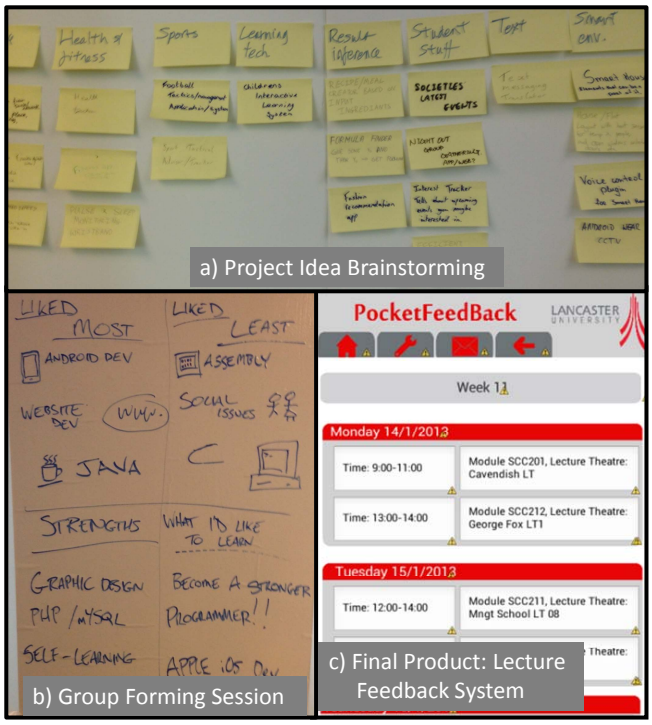

Fig. 1. Work Products from Core Studio: SCC.230

At the end of Networked Studio and Live Studio modules the students were able to deliver high quality products, some of which had commercial potential. This is based on the feedback from an industry observer whose major product is computer systems for smart living environments. The industry observer was impressed by the ideas generated by the students and the fact that such a high quality product could be delivered from scratch in a short space of time – just two months including two weeks of introducing the students to the development technology and the Scrum framework. The students also created YouTube videos of their work and Fig. 2 (see http://youtu.be/YJ7X3Y_zoJk) and Fig. 3 (see https://www.youtube.com/watch?v=pzlAcsL_wto) are the screen captures from the videos.

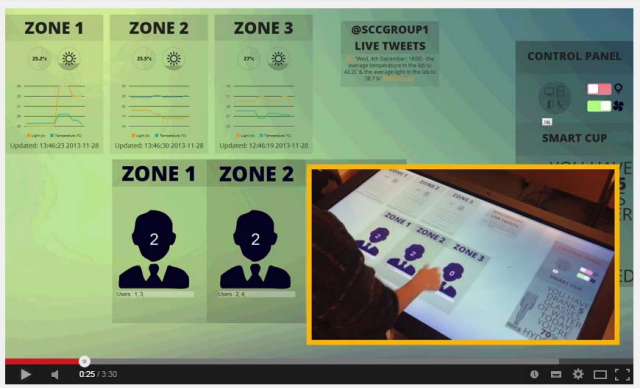

Fig. 2. Demo Video Capture of Networked Studio: SCC.330

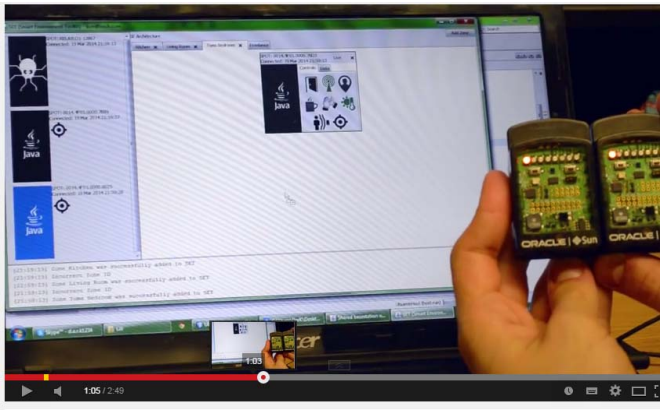

Fig. 3. Demo Video Capture of Live Studio: SCC.331

IV. Student Feedback

The Software Design Studio began in October 2012. So far, the feedback has been very good. In general students found the module as whole very good and engaged strongly with it (see Figures 4-6). In addition, module feedback on other modules run by the School highlights the positive way in which the workshop-style of the Studio is being received. For example, one student in a different degree programme, who attended SCC.230 as a part of his Generic Skills module, commented on: *'I learnt more in one 2-hour Studio session than I did in this entire module.'*

Student evaluation for all three the studio modules have been very strong. Modules are rated on a 5-point scale with highest rating being "very good" and the lowest, "poor". More than 80% of the students rated the SCC.230 Core Studio module as very good. 100% rated their involvement as very good. More than 85% of the students rated the SCC.330 Networked Studio module as very good. The SCC.331 Live Studio module was rated as very good by more than 90% of the students. Most of the students said that SCC.331 was the best module they had taken in their whole degree.

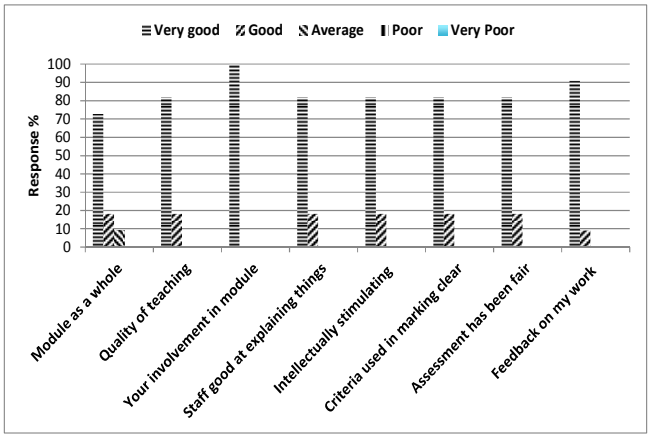

Fig. 4. Feedback on Core Studio (2012-2013)

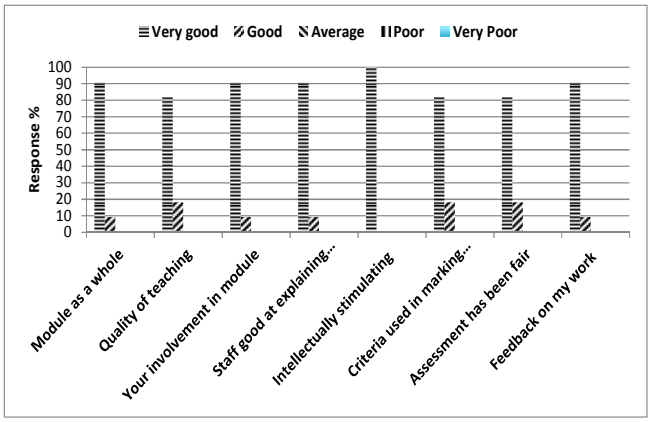

Fig. 5. Feedback on Networked Studio (2013)

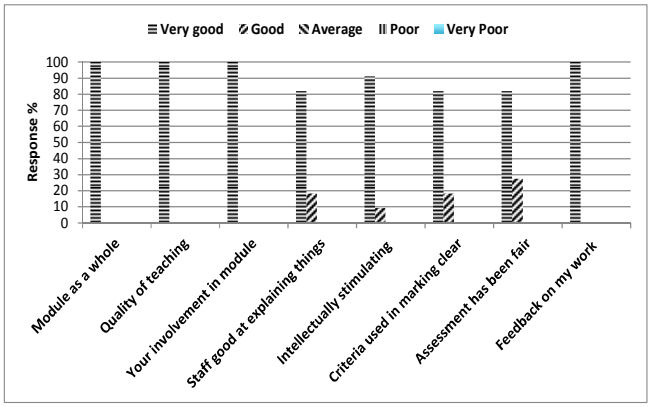

Fig. 6. Feedback on Live Studio (2014)

Student comments include:

"*The best part of the module was that we got to learn first hand from our own mistakes*", "*The approach was different to what I am used to, but proved to be very effective*";

"*Having a dedicated studio for our own use gave me a quiet place where I could focus on my studies*";

"*I was significantly more productive in there than anywhere else I tried working*";

"*Pretty much everything was useful. The whole design process was covered in great detail and I now feel as though I could complete this to a much higher standard than before*"; and

"First time I've enjoyed working in a group, having a big focus on individual work as well as group work encouraged people to put a lot of effort in."

V. LESSONS LEARNED

We had encountered a number of challenges in setting up and delivering the Software Design Studio. Along the way we have learnt several lessons. We have distilled the lessons below.

- *It is important that the students feel sense of ownership of the Studio right from the start.* The Studio's physical lab is a key element of the success of the approach. The lab is available 24 hours for students, they are told to view it as their 'home', and have complete control over how it is used. Students are responsible for maintaining the lab. They use it for dedicated work sessions as well as social aspects of the work (e.g., pizza party design sessions). Achieving this has been a challenge, however, as it required a shift in focus in how lab spaces are viewed – from the top-down, tightly controlled management style to student-led management.
- *Adopt an agile model for feedback and assessment*. Given the focus on practical hands-on work and on teamwork, it was challenging to provide detailed feedback for both the team and the individual in a timely manner. To address this, the assessment followed an agile model in which each system release was assessed weekly both in terms of team and individual contribution. Feedback was given immediately and in-situ in the workshop session. This was done by students giving weekly presentations of their work, followed by peer critique of the presentation and the code produced. In addition, weekly 'stand-up' meetings were instituted – lightning meetings where each student reviewed their progress for 5 minutes – to motivate students and check the status of deliverables.
- *Rebadge software engineering principles.* Another challenge has been how best to integrate theory and practice in the studio. Given the focus on practical and hands-on work, the teaching staff members had to work hard to ensure that students learned the theory of software engineering as well as how to build things. To address this, we introduced a 'badge' assessment scheme: students build their product but have to collect four badges – each corresponding to a particular theoretical aspect – on which they are assessed. For example, students might collect a badge on 'software design' where they do extra work on studying the theory behind software design and how it relates to their practical experience.
- *Studio approach fosters modeling*. We also have observed the role of software models in the software design studio [11]. A year and a half long observation of the studio has

revealed that, in this style of education, students make little use of formal modeling notations but heavy use of informal model sketches. Since there is strong evidence that the studio model is an effective way of teaching software engineering, we argue that these observations have implications for the way modeling is currently taught as well as how it might be taught differently.

Finally, we put a lot of thought into getting students to reflect on their practice – in an attempt to instil 'reflective practice' [12] into the curriculum. This was done through a variety of means – peer critique, constant design critique, coaching, and individual self-reflection reports. Again, this was challenging because it required a change of culture in the students, who were not used to reflecting on their practice in this way.

In addition to these successful factors, we also have identified some challenges that need to be addressed in the future.

- Scale the size of the class: we deliberately design the modules for small number of students – we expect twenty is the maximum number of students that we can manage in this current format of project management (i.e., Scrum) and the supervision with two dedicated academic staff members. This is mainly due to the resource constraints (e.g., available teaching staff, lab space, available devices, etc.);
- Train the 'trainer': the studio based teaching approach is new to the students, but also new to the teaching staff members. When we first introduced this new concept of teaching, we had to train ourselves for managing the studio workshops and Scrum, supervising students in this context, and assessing/tracking the progress of students' attainment. There were quite a few occasions that the teaching staff members attempted to switch to the traditional lecturing mode, rather than observing and coaching the students. Based on our experience, we are now more confident with the studio approach and have better understanding on how to supervise the students; and,
- More connection to industry: we had invited speakers from industry and we observed that these events could motivate the students to demonstrate their achievements to the visitors. Also the talks about real industry challenges were very valuable for the students. We would like to provide more of these occasions to the students and it might help them to establish their own relation with the industry.

VI. Related Work

Although studio-based pedagogy is increasing in software disciplines, it is acknowledged that not many have been implemented [13]. The use of software studios has been around for a couple of decades, with one of the first attempts at implementing studio education going as far back as 1990 [14]; although there are numerous publications about studio implementations, it has previously been noted that an up-to-date and exhaustive review of all studios in software engineering is not yet available [7]. A recent brief review of studio-based learning in computer science was conducted which describes five institutions that have implemented a studio course [15] (University of Queensland, Monash University, University of Victoria, Washington State University, Auburn University). Some other institutions with prominent publications also have studio implementations, these are: Carnegie Mellon University, Massachusetts Institute of Technology, University of Hawaii, Poznan University of Technology. This list does not take into consideration the HCI or media-based studios. Despite the low number of software studio courses, they often report varying levels of success. However, comparing these courses is difficult as none of them have previously followed a shared definition or understanding of studio education [6]. This is a proposed benefit of the 'studio framework,' with which studios can be compared against it, providing a foundation for comparisons.

Previous work [7] has presented the results of an ethnographic study on Lancaster's studio. The observations were made during the studio's first year (2012-2013) and were reflected against the 'studio framework' [6]. In this work it was argued that studios should not be considered a binary state (i.e. whether it is or is not a studio), due to their complex nature; there will be some spaces, which are more 'studio-like' than others, and as such, some spaces may appear to not cover all aspects presented in the studio framework. Despite this, it was concluded that the studio implementation at Lancaster satisfied all of the framework's categories, with example observations provided for each. Some noteworthy conclusions made based on the observations were the importance of obtaining 'a sizable dedicated room,' and the significant use of the mobile whiteboards by the students (e.g. for supporting impromptu collaboration, enabling the externalisation of ideas, group awareness of progress and designs, and enabling frequent critique) - although simple assets, they were used in various ways, allowing for significant coverage of the framework.

VII. Conclusions

Software engineering education remains as one of the most important area to be successful in academia and industry. Unfortunately, the teaching of software engineering still poses a number of difficulties. Unless these challenges are addressed, successful transfer of this promising technology is unlikely to be realised. This paper has described a studio based teaching approach at Lancaster University that aims to address the teaching challenges by combining the design studio approach, the agile framework Scrum, and effective project support environment.

The feedback on the Software Design Studio – from both staff and students – has been outstanding. The Bachelor of Science (BSc) in Software Engineering is quickly becoming a flagship programme of the School of Computing and Communications. Although the programme is designed as a small, elite programme there is interest to see if the teaching methods can be transferred across to the much larger BSc in Computer Science. Moreover the Studio is already becoming a unique selling point of Lancaster's computer science education; the first three authors of this paper won the Lancaster University Pilkington Teaching Award, which was awarded annually for ex-

cellence and innovation in teaching and for making a significant impact on the student learning experience.

REFERENCES

[1] G. Kotonya. and J. Lee, "Teaching reuse-driven software engineering through innovative role playing," In Proceedings of 36th ICSE (Hyderabad, India) on Education 2014. 276-282.

[2] I. Sommerville, "Software engineering," Addison-Wesley, 2010.

[3] A. Finkelstein, "Software Engineering Education: A Place in the Sun?" In Proceedings of 16th ICSE Workshop on Software Engineering Education (Sorrento, Italy, May 1994). ICSEW 1994. 358-359.

[4] L. Brodie, H. Zhou, and A Gibbons, "Steps in developing an advanced software engineering course using problem based learning," Engineering education, 3(1), 2008.

[5] C. N. Bull and J. Whittle, "Supporting Reflective Practice in Software Engineering Education through a Studio-Based Approach," In: *IEEE Software* 31.4, pp. 44-50

[6] C. N. Bull, et al., "Studios in Software Engineering Education: Towards an Evaluable Model," In Proceedings of the 35th International Conference on Software Engineering (ICSE '13), IEEE Press (San Francisco, CA, USA) 2013. 1063-1072.

[7] C. N. Bull and J. Whittle, "Observations of a Software Engineering Studio: Reflecting with the Studio Framework," In Proceedings of the 27th Conference on Software Engineering Education and Traning (CSEE&T '14), IEEE Press (Klagenfurt, Austria) 2014. 74-83.

[8] D. Tyman, N. Bulusu and J. Mache, "An activity-based sensor networks course for undergraduates with sun spot devices," In Proceedings of 40th ACM technical symposium on Computer Science Education (SIGCSE'09). New York, NY, USA: ACM, pp. 34-38.

[9] R. Pichler, Agile Product Management with Scrum: Creating Products that Customers Love, Addison Wesley, 2010

[10] A.M. Davis, "Fifteen principles of software engineering," IEEE Software, 11(6), 1994, 94-96.

[11] J. Whittle, C.N. Bull, J. Lee, and G. Kotonya, "Teaching in a Software Design Studio: Implications for Modeling Education,' Educators Symposium at MODELS 2014, 2014.

[12] Schön, Donald A. (1987). Educating the Reflective Practitioner. San Francisco, CA: Jossey-Bass

[13] C.D. Hundhausen, N. H. Narayanan and M.E. Crosby, "Exploring studio-based instructional models for computing education," In Proceedings of the 39th SIGCSE technical symposium on Computer science education (SIGCSE '08). New York, NY, USA: ACM, pp. 392–396.

[14] J.E. Tomayko, "Teaching software development in a studio environment," ACM SIGCSE Bulletin 23.1, pp. 300–303.

[15] A. Carter and C. Hundhausen, "A review of studio-based learning in computer science," J. Comput. Sci. Coll., vol. 27, no. 1, pp. 105-111, Oct, 2011.