A Design Studio Course in Application Development: Lessons Learned

Steven R. Haynes & David R. Mudgett
College of Information Sciences and Technology
The Pennsylvania State University
University Park, PA, USA

***Abstract*—Degree programs in practice-based disciplines such as architecture, industrial engineering, graphic design, and the fine arts often include design studio experiences for their students. These studio courses are intended to provide students with an opportunity to practice skills and apply knowledge gained from curriculum courses following a more traditional format. They are also intended to provide a forum for both peer-to-peer and instructor-student communication and critique around student work. In this paper we report on lessons learned from the first three years of a design studio program in the College of IST at Penn State University. We provide a short overview of the design studio concept and explain how the idea was adapted to the domain of modern application design and development. The paper is intended as an experience report, giving the rationale and initial design for the course, outlining lessons learned in delivery, and describing some of the experimentation and redesigns carried out to mitigate identified challenges and leverage emerging opportunities.**

Keywords—design studio; application development; information sciences and technology.

I. INTRODUCTION

For the last three years we have been offering two design studio courses for undergraduate students in Information Sciences and Technology (IST). The rationale behind these courses was that IST students should have an opportunity to practice application design and development every semester of their undergraduate career, and have a forum where they can interact with other application developers to learn from each other's work. The studio courses have been popular and are generally considered to be a success, though we have also experienced some significant challenges attempting to recreate the studio as they are delivered in the arts and in architecture.

In this paper we describe the rationale for proposing and developing the two courses, which are very similar, describe their structure, and report on some of what we have found to be effective and ineffective in a course of this type. Studio courses have the potential to provide students with an important learning experience where they can apply concepts, techniques, and tools learned in more traditional courses, but there are many challenges.

The rest of the paper proceeds as follows. In the next section we provide some background on the design studio concept and its roots in arts and architecture. We then review some of the ways the studio concept has been adopted and adapted within fields related to computing and communications technology. We outline our own course design, and then report some of the experiences and lessons learned from delivering the course over the last three years to approximately 400 students. We conclude with some of the changes and innovations we have planned moving forward.

II. THE DESIGN STUDIO CONCEPT

The concept of a design studio is based on the idea that students in a particular discipline should be given opportunities to practice skills first learned in more conventional course formats, such as those employing lecture, or problem-based pedagogies. Studio courses typically emphasize individual learning over teaching, and include few if any lectures or other structured content delivery activities. The studio course format has long been used in the visual and physical arts, in performance-based arts such as dance and music, and in architecture. Though the status of the studio in traditional universities is sometimes called into question [1], it is interesting to note that origins of the word studio lie in the Latin word *studeo*, to study. In fields where designing and making characterize activity, some suggest that most deep learning only occurs when students are actively engaged in applying what they know to realistic problems [2].

One study of examining perceptions of the design studio experience [3] found several factors that seem to influence the effectiveness of the approach among them that learners can switch freely between different learning approaches; that collaboration with others promotes inspiration and reflection; and the positive culture of critique germane to the studio format.

In addition to practice, the activities most central to the studio course format are coaching and peer-to-peer critiques, or 'crits'. In a painting or pottery studio these can be conducted with either the instructor or student reviewer simply walking around the studio space and offering guidance and constructive criticism of students work. Positive and constructive critiques contribute to a culture of self-reflection by causing the critiqued student to reconsider design decisions made in isolation [4].

The term design studio may be used to refer to a physical space and the importance of an appropriate space for the

activities being performed has been highlighted in the literature [5]. University colleges and departments in the arts and architecture typically include purpose-built space for students to practice their skills, such as an auditorium with a stage or a well-lighted room with drafting tables. The presence of a dedicated space can contribute to the feeling that students are working in a simulated professional environment rather than a classroom.

One of the goals of the studio course format is to go beyond teaching students what design is and to help them learn to create good designs [6]. This kind of learning comes from the iterative cycle of design, build, and co-evaluation that commonly occurs within the studio environment. The studio format inherently entails active learning as students work through the "conversation with materials" at the heart of reflective practice [7].

Design studios are considered a critically important "bridge" between knowledge and skills acquired in the academic context and their application in a professional context [8]. In the studio students are given the freedom to practice these skills in a relatively low-stakes environment, where no one gets fired, and where the student can gain confidence in their ability to find solutions to problems of realistic scale and complexity [9]. Similarly, the co-production of knowledge resulting from instructor and student-peer critiques has many parallels with the kind of learning that regularly occurs in the practical context.

III. Studios in Computing and Software Development

Mitch Kapor, founder of Lotus Development Corporation and creator of Lotus 1-2-3, was among the first to propose that software developers could fruitfully mine practices from the field of architectural design [10]. He argued that software developers should reframe their perspective from a focus on the purely technical and engineering aspects of coding problems to one that considers first user needs and the usability of the artifacts that result from development. He called out a particular need for software development training to include a studio component.

> *"Most important, students learn software design by practicing it. A major component of the professional training, therefore, would consist of design studios in which students carry out directed projects to design parts of actual programs, whole programs, and groups of programs using the tools and techniques of their trade."*

Reflecting on the studio course taught by Kapor at MIT, Kuhn [11] argues for a software studio format to include among its central components:

1. Student projects that address complex and open-ended problems.
2. A "culture of critique" where instructors and students are tasked with providing constructive feedback on the work performed in the studio.
3. Fostering an appreciation for the range of issues that can both enable and constrain the software design process including those from the technical, psychological, organizational, social, and regulatory domains, to name only a few.
4. Consideration of successful and unsuccessful precedents (patterns) as solutions to problems that recur across different software development contexts.
5. A holistic perspective on the role of software in modern life including all of the varied factors that can potentially impact the usability and usefulness, and therefore the design, of an application.
6. Methods to help manage the scope of the in-studio design process given the conflict between a desire for 'real world' problem solving and the constraints of the university semester or quarter system.
7. The importance of tools and other "design media" in influencing the design and development process.

Both Kapor and Kuhn did their work in the 1990s, when the internet was just emerging and society had not yet adopted computing and communication technologies as an integral part of their lives. The pervasiveness of these technologies and of software applications makes these lessons all the more important for today's students of software design and development.

Studio courses are often concerned with providing opportunities for students to practice use of tools, techniques, methods, and concepts learned in more traditional classroom settings. The studio format may involve the use of field work, where students are asked to engage with 'real' prospective users in the design and assessment of their applications [12].

One aspect of design problem solving often covered in the studio format is the process of identifying and selecting among possible alternative solutions to the same set of functional requirements [13]. This and other approaches intended to reduce solution-first thinking are often ignored in more traditional classroom settings where the constraints of a course topic load prevents consideration of how and why different techniques and tools can be applied to similar problems.

The status of design and construction as knowledge-generating activities is widely debated and still uncertain [14]. Seymour Papert [15] famously declared that people, in this case especially children, learn best while engaged in the process of making something. Schon [1] points out however that the activities central to design and construction bear a close resemblance to what scientists actually do, even if their motivations and products differ considerably.

Human-computer interaction may be the obvious place where the studio approach can be usefully applied, especially given the field's shift in focus from the psychology of HCI to designing and making more usable computing applications [6, 16]. The influential design firm IDEO has been a vocal proponent of a more studio-based approach to interaction design [17], one which values creativity and innovation of more purely psychological or engineering-based perspectives. The emergence of *hackerspaces* [18] and *hackathons* [19] is evidence that this approach to studio thinking is gaining ground in both practice and academia.

The studio format has also been applied to support software engineering courses. Woodley and Kamin [20], for example, argue that software engineering and software development requires that students be given opportunities to practice the kind of iterative refinement supported by feedback and characteristic of this kind of work in industry. They claim that large class sizes and other resource constraints otherwise prevent this critical form of learning and professional development.

Among the many challenges that have been identified with migrating the studio concept to 'non-native' fields is that students unfamiliar with the approach may experience uncertainty and anxiety regarding expectations and grading rubrics and how they will be applied to their work and work products [13]. More senior students in particular might suffer as they often grow accustomed to the "rhythm" of conventional lecture and laboratory courses and how their performance in these is assessed.

In one case where the studio format was used for learning software engineering, Bull, et al. [3] found that achieving all of the benefits of the studio required creating a culture akin to that found in arts and architecture. Part of the problem is that in arts and architecture the studio component typically involves a much more significant portion of the students' time than those that have been adapted to software and other technology design domains. Another finding from this work was that students were typically not provided with the dedicated work space usually associated with an arts and architecture studio.

Lee, et al [21] provide a detailed account of their motivations, design process, and preliminary results from delivery of a "software studio" course. This work provides a useful template for others considering development of a studio experience in software development. They also report early results and lessons learned from delivery of the course, which give much useful guidance for new studio instructors. We hope the experiences reported later in this paper can also be used this way.

IV. IST'S APPLICATION DEVELOPMENT STUDIOS

Since 2013 we have offered two application design and development studios in the College of Information Sciences and Technology (IST) at Penn State University. The decision to propose, design, and deliver these courses had a number of motivations. First, we felt that students would benefit from a course where they could apply knowledge and skills learned in their more traditional courses in a self-directed environment on projects of their own choosing. Second, over the years students had requested courses in a very broad range of systems development languages, platforms, and technologies, and we believed that the studio format would provide an opportunity for them to select and explore these on their own, with instructor and student peers providing coaching as needed. Finally, we wanted to ensure that students could include a design and development course in each semester of their undergraduate experience. The breadth of the IST curriculum is a strength of the program, but as a consequence students in the major often go one or more semesters without a course that includes programming as a significant component. We wanted to eliminate these gaps so that students could practice application design and development in one form or another continuously through graduation.

We proposed and designed two studio courses for IST's Design and Development (D&D) option. The first is targeted at second/third-year students who have completed their first two application development courses, which are roughly analogous to the first two programming courses in a typical computer science curriculum but with an emphasis on application development. At this early point in their academic careers students are able to create a basic, standalone application using the Java programming language and its various libraries for creating graphical user interfaces. This first studio provides students a forum where they can practice what they have learned in their first two courses by applying their new skills to creation of a basic but fully functional Java application. We use Java as the base language of instruction in our design and development option.

Our overarching curricular goal is that after two taught programming courses and the first design studio, students will have intermediate-level skills in the Java language and the ability to create a standalone, graphical, event-driven application of medium complexity that performs some useful task for one or more identified users. On entry to this first studio course, most students have studied procedural and object-oriented programming (a two semester sequence), but have not yet learned object-oriented design, project planning and management, user requirements analysis, software development lifecycle models, and other abstract concepts of system design and software engineering. Therefore, another goal of the course is to provide a hands-on practicum where students encounter these topics *in context* before studying them formally. The intent is that this early hands-on exposure will both motivate and help students understand these important ideas at a practical level before studying their more abstract aspects in upper-division courses.

The second studio course is intended for third/fourth-year students who have completed at least three courses in the application design and development curriculum including object-oriented design and development. Some students in the upper-level studio have completed up to five development courses plus all of their other core technical requirements. In this second studio students are given more freedom to choose programming languages and development platforms from across the spectrum of modern practice. Also, students in the second studio often choose to develop applications for the web, for mobile devices, and for the internet of things. Students in this second studio are encouraged to gain skills and practical knowledge in areas that will directly support their career ambitions.

The courses as proposed and first delivered both followed the same overall approach, though there have been a number of modifications as described in the next section on lessons learned. Because the focus in IST is on application design and development rather than pure programming, both were designed to include activities from across the systems development lifecycle, though there was initially no intended commitment to agile, waterfall or other specific development

methodologies. Key competencies covered in prior courses and expected to be exercised on studio projects included:

- Application Project Planning – Gannt charts, PERT chats, functional decomposition, activity allocation, skill mapping.
- Domain Analysis & Problem Structuring - stakeholder analysis, personas, apprenticing with the user, formal and informal problem structuring methods.
- Application Requirements Analysis – scenario-based methods, use cases, task analysis, utility-theoretic design.
- Application Design – activity modeling, object-sequence modeling, class diagrams, statecharts, wireframing and other low-fidelity prototyping methods.
- Application Development – programming using application programming interfaces and other source code resources, applying design and architecture patterns, integrated development environments.
- Application Quality Assurance – black and white box testing; unit, integration, and system testing, test cases, scripting and other testing tools.

Students propose their own ideas as projects for the studios and a project proposal and justification is typically the first deliverable. The proposal forms the basis for a discussion between student and instructor about what the student wants to achieve in the studio, including skills they feel they need to practice and new areas they want to explore. Project proposals related to students' deliverables in other courses are considered acceptable, but require disclosure by the student and the agreement of studio instructors. Initially it was intended that projects could be undertaken by individuals, pairs, or in larger groups with each studio participant responsible for producing significant individual project deliverables. The proposal phase of the studio is also an opportunity for the instructor to provide feedback on the scope and depth and of the proposed project to ensure that a working piece of software results from their work over the semester.

The general structure of both the second and third year studios is similar, with expectations of deliverable sophistication being the primary differentiator. Both follow a schedule corresponding roughly to that given in the table below.

TABLE I. APPLICATION DEVELOPMENT STUDIO WEEKLY SCHEDULE

Week #	Activity/Deliverable
1	Project Proposal
2	Concpetual Design
3	Product Backlog & Plan
5, 7, 9, 11, 13, 15	Development Sprints

Following examples from arts and architecture [12], first versions of the studios required students to maintain a design and development journal. This journal is intended to be the analog of an engineering notebook or artist's sketchbook and should contain a running account of the students design and development ideas, explorations, rationale, and other notes. The design journal is one of the assessed deliverables and an important marker of the students' effort and progress. Part of the rationale for the journal component was to promote design thinking, including an appreciation for design rationale and the processes of problem structuring, solution development, and consideration of design alternatives and their trade-offs.

In addition to project deliverables and design journals, peer and instructor design critiques were proposed as feedback and assessment mechanisms for the studio courses. Collaborative learning and instructor/peer critiques are central to the studio learning experience. It was initially planned that this aspect of the courses would be supported by scheduling student peer review sessions after completion of each significant project deliverable, proposals, designs, and successive versions of the application build. Instructor critique was planned as an ongoing process with continuous interaction between instructor and both individual students and project groups.

The IST design and development studios were designed to provide a forum for students to actively engage with the concepts, process, tools, and materials used to envision and build software applications. In particular our intent was to emphasize experimentation and risk-taking, and to help build enthusiasm for the process of designing and building working software applications. Successful participation in at least one of the studios is now a graduation requirement for all IST students in the Application Design & Development option. In the next section we describe the different ways that the studios have evolved in response to our experiences.

V. LESSONS LEARNED

In this section we report our experiences offering the two IST studio courses over the last six semesters. Approximately 400 IST students have participated in one or both of the studios. Introduction of the studio component as an integral part of the IST Design and Development option is generally considered a success. Student reports are generally positive, and there are many cases where interesting working software applications have resulted from student work in the course. Nonetheless it has become clear to us (the authors) as instructors for the studios that some aspects of the course design are effective, but also that there are many challenges to achieving the initial vision for the courses.

Among the significant challenges faced by students in the course are defining the scope and scale of a 15-week application development project, conducting project estimation, managing application integration and interoperability, development platform challenges, and working independently to reliably meet project deadlines. These challenges point to some of the unique and important learning opportunities that may only arise in the design studio course format. We detail both positive and negative aspects below, and include some ideas (mitigations) for how our

experiences can be used by others wishing to include an application design studio as part of their undergraduate curriculum.

A. Project Scoping

One of the first issues we identified was that students often have unrealistic project ambitions for their studio experience. This has resulted in a number of on-course problems, but also presents an important learning opportunity. Our semesters are 15 weeks, which as any professional application developer would attest, is really not sufficient time to create a polished piece of application software of any complexity. Projects of unrealistic scope and scale can have a cascading negative impact on a student's studio experience as they realize, often fairly quickly, that they must re-think their commitment to their project of first choice. In the worst cases, students petitioned to change projects altogether, a move that only served to exacerbate the problem of a short semester timeline. The effect on student morale was often evident as they struggled to develop a new project idea within the short time available.

After the first semester delivering the studio, subsequent semester saw an increased focus on scoping of the student proposals along with more discussion between instructor and student regarding the viability of their planned application. Increased emphasis on use of the Scrum method, of identifying small, discrete 'chunks' of functionality that can implemented within a two-week sprint, has been especially helpful. Students are advised to carefully consider the idea of a minimal viable product (MVP) and then work backwards to identify the critical features they will need to achieve this level of functionality. Despite some of the negative effects of mis-scoping projects, we consider this an effective learning opportunity as students begin to develop skills in software scoping and estimation.

B. Project Estimation

Project estimation is closely related to project scoping, as discussed above. One reason project scoping is difficult for these studio students is that, especially in the first studio, they have virtually no background in software project management and estimation. Software project estimation is difficult, even for seasoned professionals. These skills are essential to set realistic goals and expectations, but software project management and estimation techniques are taught later in the program as part of a software engineering course. It would be advantageous to cover project planning and estimation earlier, but the curriculum is already full teaching the basic programming and design principles, as well as the many other required courses in important foundational topics such as networking, data organization, user interaction with information systems, and mathematics.

We initially planned to bootstrap some basic software project management methods early in the course, but initially underestimated the difficulty students would have coming up with even order-of-magnitude software time estimates. After the first few studio courses, we began to include the basics of software project estimation as an early discussion. Some instructors use size-based estimation such as function points or simply lines of source code (LOC), while others use story points as suggested by agile/scrum methods. In all cases, the objective is for students to develop only rough estimates to keep project scope reasonable and to help track progress to goals, not to set yardsticks by which to measure performance. Instead we measure performance by observing the development process students follow via the design journal and daily class observation, instructor and peer critiques, and the quality of regular software deliverables. As with project scoping, student experiences with estimation can appear painful but provide them with an important opportunity for individual practice and personal productivity 'calibration'.

C. Conceptual Design

By the time students reach the studio courses, they have spent considerable time learning to write Java source code to solve specific, small-scale problems. By the second studio course, they have also learned something about the software application design process, how to write use cases, how to model interactions with users, and visualize how pieces of a system fit together using different UML diagrams. Still, many students are not yet prepared to think about the larger context of a software system from the top down, or how to decompose a larger system into manageable units that they can implement unit-by-unit. These fundamentals of design thinking are among the main points of the design studios – to approach the design process more holistically.

In the second of the two studios we have created and introduced a self-paced, conceptual design tutorial that covers the basics of scenario-based design, use cases and user stories, and conceptual class diagramming in the UML. Student feedback suggests that this tutorial is useful as a tool to provide students with a short guide to best practices in early design thinking. An additional mitigation to be added is peer reviews of these early design materials. So far peer reviews have focused on working software products, but design representations such as these, which are meant to facilitate communication, will likely benefit from both peer and instructor-led critiques.

D. Project Focus

Unlike a studio in painting or ceramics, where students can easily be observed at work on their creations, we have sometimes found it a challenge to maintain students' focus on the work germane to the course, rather than using the time in class as a kind of study hall. Part of the problem may relate to the size of the studio classes, which typically range from 25 to 50 students. In an arts or architecture studio course sizes are typically much smaller, 12-20 students [6], which provides the instructor with considerable more one-on-one time with individual students. We sometimes have smaller sections, and have observed that this is less of an issue.

But another part of the problem may relate to differences between the culture of technical development and the culture of arts and architecture. The studio concept is very much a part of the practice and history of the latter, while migrations to more technical fields have only happened in recent years. Much of the work in arts and architecture is intended to be shown, while application are intended to be used (of course buildings are

also intended for use). The art portion of an interaction design is often subservient to the flow of the application in use. This suggests some different criteria may apply when student peers and instructors conduct a 'crit' of a piece of working software. Beyond the reviewer's aesthetic response to an application is the sense that the software provides something useful. In future versions of the studio we intend to make utility or usefulness an explicit and integral component of the review process.

E. Problem Solving, Time-On-Task & Perseverance

One of the most significant challenges we face both in the studio and in more traditional format technical courses is preparing students for the amount of time-on-task and perseverance it takes to build quality software. There are a number of issues to overcome here including perceptions that some students are simply better suited to programming, that they possess some sort of magic that makes programming easier for them. We work hard to impart to our students the idea that programming and application development is a practice-based activity and that being a good developer is much like becoming a good tennis player or guitarist, in that it requires significant time-on-task (practice).

We have observed that, by prior training and/or temperament, some students adapt more quickly and easily to the design and development process. But more often than not, our experience suggests that those students who are most successful are those that invest the most time on their project, both in and out of the classroom. Moving forward we want to understand how and why some students possess the motivation to make this investment so that we can help others develop similar attitudes towards the project work.

A closely related issue relates to student attitudes towards problem-solving. Often we see that students will encounter a difficult problem in their project. They look for solutions on the web, from other students, and from the instructor. When a solution is not found within a relatively short period of time they can become frustrated, often giving up on implementing the feature that is giving them trouble. Again, we spend considerable time working to help them understand that these problems are exactly what makes application development so important and application developers so much in demand. As with motivation for investing time-on-task, we continue to seek ways to help students develop a resilient attitude towards problem solving.

F. Group and Individual Work

We found early on that many of the problems inherent in group work for more traditional course formats persisted in the studio context. Chief among these is the social loafing and partitioning of responsibility that occurs when students try to optimize their workload and stay within their comfort zones, i.e., programming, visual design, conceptual design, and writing. Because we are concerned that all students practice all aspects of application development, and because our major includes substantial opportunities for group work in other development courses, we decided to eliminate group projects after the first semester the studio was delivered. In cases where students insist on the advantages of working on the same project they are asked to divide the project into two or more discrete components where interoperability becomes one more important design criteria. In general we consider the move away from group projects to have had positive results.

G. Assessment

Assessment presents a special challenge in the studio course format as the very idea of the studio is predicated on self-directed and self-motivated learning. The role of the studio instructor is meant to be that of coach or mentor rather than the person responsible for summative assessment of the students' work. Nonetheless studio are courses for-credit and so among the instructor's responsibilities is assessment and grading of student activities and deliverables. As stated earlier, we measure performance by observing the development process students follow via the design journal and daily class observation, instructor and peer critiques, and the quality of regular software deliverables.

Introduction of student software submissions as narrated screencasts has significantly improved the assessment process as it allows more time for the instructor to focus on what has actually been accomplished rather than on setup and configuration of runtime environments for student projects. Assessment in the studio context is perhaps more problematic than in more traditional course settings as the instructor must be sensitive to the differing skills and experiences of studio participants, while at the same time ensuring fairness and promoting the notion that commitment and hard work will result in higher marks.

H. Peer critiques

Peer reviews are an important part of the feedback and assessment process in any studio-format course. In our experience, peer critiques come with both advantages and challenges. Among the advantages is that students appear to be more relaxed and therefore receptive to constructive criticism coming from a fellow student. Another is that students critiquing each other contributes to the sense of community in the studio. In addition, our observations suggest that a significant component of student learning in the studio comes from assessing the work of others. They learn to recognize problems in their own work by recognizing them in the work of others.

One issue encountered in early versions of the studio course was the problem of ensuring that peer critiques were guided enough to be comparable for grading, while at the same time still allowing latitude for student innovations in the feedback the provided to peers. These first peer critiques were given as unstructured assignments, without very much guidance on the breadth and depth of coverage necessary to achieve high marks on the assignment. Critiques done in this model ranged from excellent to superficial. Subsequent versions of the course have included peer critique assignments with both a minimum word count requirement and a rubric with points assigned for feedback given on specific aspects of the project. Results from this revision have been encouraging in terms of being more consistent and of higher quality.

Some student feedback from the initial versions of the studio suggested they wanted more opportunities for peer

review and critique. In response to this we have introduced peer critiques of every deliverable produced over the course of the semester, around nine overall. Because the peer reviews are randomly assigned and each student does two per deliverable, we have found this increased focus on examining each other's work has led to corresponding increases in the feeling of community evident in student interactions. It has also increased the overall amount of feedback students receive to approximately three times that of the first version of the course.

I. Infrastructure Challenges

There are several infrastructure challenges to doing networked, web, and mobile projects in the classroom-based studio. All classroom computers are very tightly controlled in terms of permissions and open ports. Inter-computer communication is impossible except via standard ports like including http and sftp. Installed software must be planned for and completed before the start of the semester, making on-the-fly installs to support student projects impossible. Therefore, students who want to use any software not in the standard install or communicate with other computers generally need to provide their own equipment. This is not usually a serious problem – our design and development option students tend to be fairly technology-savvy and are naturally interested in the technology. We have had students do projects using many platforms, including iOS, Android, Arduino, Raspberry Pi, many flavors of Linux, and the usual Mac and Windows platforms.

Where this issue has become problematic is when students share their work-in-progress with the instructor or their peers for either critique or assessment. Student development and runtime environments can be complex and it has become unrealistic for the instructor to have all of the infrastructure necessary to run any student project. This has resulted in cases where too much time was spent creating environments rather than reviewing the essential elements of a student's work.

We have recently begun requesting that students submit their work for critique or assessment in the form of a screencast video. In these videos the students are instructed to provide a three part demonstration. Part one focuses on an overview of work and progress completed since their last submission. This generally covers one two-week development sprint. The second part is a user-facing demonstration of the usable capabilities now provided by their project. This is generally one or more user stories. The third part is source code walkthrough and review. In this third part students explain how and why they have implemented certain capabilities in certain ways. Student peer reviewers use both the submitted screencasts and face-to-face meeting to conduct their peer reviews. The instructors use them to complete assessments of student progress.

The video screencast approach to peer-review and submission for assessment has proven successful in eliminating much of the infrastructure setup and configuration work previously required to review student work. The first attempt at the approach limited screencasts to five minutes, which has proven too short for the format. Screencasts in future will have an increased time limit of ten minutes.

VI. CONCLUSION

Studio courses are a kind of mass independent study, where students work independently but in close contact with other students carrying out similar kinds of work. Our brief review of the literature and history of the studio concept suggests that achieving a certain cultural identity, in our case, a community of application development practice, is critical to a successful studio experience. Workspaces, interactions between students and between students and instructor-coaches, demonstrations, and criticism and critique all contribute to developing and sustaining this culture. This report describes our first steps in attempting to achieve such a community within an undergraduate program in application design and development.

As Schon [1] has pointed out, the role of a studio course is sometimes at odds with the stated mission of a traditional, research-focused university. As other have found, achieving a close mapping between the arts and architecture studio model and one in computing and information technology design presents a number of special challenges [6]. It is likely the case that a new model of studio-based learning is required for application development, one that takes into account both the opportunities and constraints presented by working within and on digital media.

Overall, our experiences designing and delivery application development studio courses have been positive. Student feedback on the courses is generally positive and constructive, and we have adopted many of the suggestions given through both formal and informal feedback mechanisms. Though the design studio format of instruction and learning presents some challenges when applied to the application development domain, we believe it provides some unique learning opportunities not otherwise provided in more conventional course formats.

ACKNOWLEDGMENT

The authors would like to thank our colleagues in the College of Information Sciences & Technology for supporting our work on both new courses and curriculum redesign.

REFERENCES

[1] D. A. Schön, "Toward a marriage of artistry & applied science in the architectural design studio," *Journal of Architectural Education,* vol. 41, pp. 4-10, 1988.

[2] H. A. Simon, *The sciences of the artificial*: MIT press, 1996.

[3] C. N. Bull, J. Whittle, and L. Cruickshank, "Studios in software engineering education: towards an evaluable model," in *Proceedings of the 2013 International Conference on Software Engineering*, 2013, pp. 1063-1072.

[4] C. N. Bull and J. Whittle, "Supporting reflective practice in software engineering education through a studio-based approach," *IEEE software,* pp. 44-50, 2014.

[5] D. Vyas, G. van der Veer, and A. Nijholt, "Creative practices in the design studio culture: collaboration and communication," *Cognition, Technology & Work,* vol. 15, pp. 415-443, 2013.

[6] Y. J. Reimer and S. A. Douglas, "Teaching HCI design with the studio approach," *Computer science education,* vol. 13, pp. 191-205, 2003.

[7] D. A. Schön, *The reflective practitioner: How professionals think in action* vol. 5126: Basic books, 1983.

[8] C. B. Brandt, K. Cennamo, S. Douglas, M. Vernon, M. McGrath, and Y. Reimer, "A theoretical framework for the studio as a learning environment," *International Journal of Technology and Design Education,* vol. 23, pp. 329-348, 2013.

[9] J. Prior, A. Connor, and J. Leaney, "Things coming together: learning experiences in a software studio," in *Proceedings of the 2014 conference on Innovation & technology in computer science education*, 2014, pp. pp. 129-134.

[10] M. Kapor, "A software design manifesto," *Dr. Dobb's Journal,* vol. 16, pp. 62-67, 1990.

[11] S. Kuhn, "The software design studio: An exploration," *Software, IEEE,* vol. 15, pp. 65-71, 1998.

[12] J. Mathews, "Using a studio-based pedagogy to engage students in the design of mobile-based media," *English Teaching,* vol. 9, p. 87, 2010.

[13] P. Little and M. Cardenas, "Use of “studio” methods in the introductory engineering design curriculum," *Journal of Engineering Education,* vol. 90, pp. 309-318, 2001.

[14] D. C. Edelson, "Design research: what we learn when we engage in design," *The journal of the learning sciences,* vol. 11, pp. 105-121, 2002.

[15] S. Papert and I. Harel, "Situating constructionism," *Constructionism,* vol. 36, pp. 1-11, 1991.

[16] E. Blevis, Y.-k. Lim, E. Stolterman, T. V. Wolf, and K. Sato, "Supporting design studio culture in HCI," in *CHI'07 Extended Abstracts on Human Factors in Computing Systems*, 2007, pp. 2821-2824.

[17] P. Spreenberg, G. Salomon, and P. Joe, "Interaction design at IDEO product development," in *Conference Companion on Human Factors in Computing Systems*, 1995, pp. 164-165.

[18] S. Lindtner, G. D. Hertz, and P. Dourish, "Emerging sites of HCI innovation: hackerspaces, hardware startups & incubators," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 2014, pp. 439-448.

[19] G. Briscoe and C. Mulligan, "Digital innovation: The hackathon phenomenon," *London: Creativeworks London Work Paper,* 2014.

[20] M. Woodley and S. N. Kamin, "Programming studio: a course for improving programming skills in undergraduates," in *ACM SIGCSE Bulletin*, 2007, pp. 531-535.

[21] J. Lee, G. Kotonya, J. Whittle, and C. Bull, "Software design studio: a practical example," in *Proceedings of the 37th International Conference on Software Engineering-Volume 2*, 2015, pp. 389-397.