Towards an industry-collaborative, reflective software learning and development environment

Julia Prior, Chanissara Arjpru and John Leaney
Faculty of Engineering and Information Technology
University of Technology, Sydney
Sydney, Australia

***Abstract*— A significant mismatch (88%) has been found between what employers and graduates perceived as important abilities and how universities had prepared graduates for employment. Conventional Teaching and Learning approaches fall short of providing the kind of learning experiences needed to prepare graduates for the realities of professional practice in industry. On the other hand, current students have very different learning styles than their forebears. Their learning preferences are experiential, working in teams, and using technology for learning. One solution to address this mismatch issue is the software development studio. Our aim is to provide an industry-collaborative, reflective learning environment that will effect the student's development of holistic skills, such as teamwork, collaboration and communication, together with technical skills, in a discipline context. This paper further describes the design and validation via prototyping for our software development studio, the progress that we have made so far, and presents the preliminary insights gleaned from our studio prototyping. The prototypes raised issues of attitudinal change, communication, reflection, sharing, mentoring, use of process, 'doing time', relationships and innovation.**

***Keywords*— software education, software engineering, software development, software factory, software studio, reflective practice, pedagogy.**

I. INTRODUCTION

In the Information and Communication Technologies (ICT) discipline scoping project [4], a significant mismatch (88%) was found between what employers and graduates perceived as important abilities and how universities had prepared graduates for employment. Employers expressed dissatisfaction with personal and interpersonal skills of graduate recruits. Graduates suggested a number of improvements, including interactive sessions, real-world examples, group work related to industry practice and meaningful problem-solving activities.

These concerns ought to impact on the learning environments provided for Software Engineering and ICT (SE/ICT) education. Addressing these concerns by using non-traditional approaches could also provide leverage for a graduate entering the SE/ICT profession. Such approaches are based on active learning models, addressing learning as more discursive and collaborative. An example of this type of approach is the use of studios [2]. As the Studio Teaching Project [7] demonstrates, there is a long tradition in the use of this approach in the creative arts disciplines, which is firmly based in Schön's work on the reflective practitioner[6][34]. Over the last twenty years, the studio approach has gained traction in the ICT and Engineering disciplines because it offers a superior learning experience, especially in achieving practical skills [35][19][15]. A famous application, where the studio is used for 40% of the Masters' degree in software engineering and has run for 23 years, is at Carnegie Mellon [33] [19]. In undergraduate education, studios are more often applied to capstone courses rather than foundational learning [31], although increasing activity at other levels is being reported[9]. [1]

In Australia, enrolments in information technology (IT) degree courses dropped by almost 20% per annum between 2002 and 2008 [37], and there has not been a noticeable reverse in this trend in recent years. There has been a steady decline in domestic enrolments in undergraduate IT degrees over the last decade: the number of tertiary ICT graduates in 2012 was half of the 9,093 who graduated in 2003 [1]. We imagine that similar figures apply to Australian software engineering degrees, but no quantitative information is available.

The lack of interest by students in the SE/ICT profession appears to arise from, firstly, a lack of appreciation of the software profession [4], and, secondly, current practices in higher education not matching prospective students ways of working [2]. Conventional Teaching and Learning approaches fall short of providing the kind of learning experiences needed to prepare graduates for the realities of professional practice in industry [10] [5]. Therefore there is a need for innovative approaches to SE/ICT undergraduate teaching. The characteristics of the studio environment [11] suggest studio-based approaches in SE/ICT education are a potential solution for addressing these problems and will help meet industry's increased expectations of graduates' work-readiness.

[1] The beginning of the introduction was formulated during discussion with Dr Jocelyn Armarego in 2013

II. Related Work

Several researchers have cited the recent problems of software engineering pedagogy. Stiller & LeBlanc claim that how we teach software engineering can have an effect on whether software development is well practiced in industry [36]. According to Garg & Varma, inadequate software engineering education is due to insufficient interaction with industry and lack of experience as software engineering educators[20]. The major challenge of teaching software engineering is to integrate applied methodology and theory into the practice of software development [28][29].

Armarego [2] shows that current students have very different learning styles than their forebears. Their learning preferences are: experiential (learning in projects, 'learning by doing'), working in teams, and using technology-based learning. Stiller & LeBlanc suggest that students will not be convinced of the benefits of software engineering techniques until they experience the benefits themselves [36]. These authors suggest further that the most effective way of convincing students that software engineering is critical to their professional development is to complete a semester-long project. There is also a need for a collaborative teaching and learning environment in teaching software engineering. One study has shown that a collaborative environment better prepares the African-American Millennial students in IT for the real world; further, it is likely to better prepare all IT students for the professional practice [40][25]. Proposed aspects of professional software development practice to include in tertiary courses are: team-based development, problem definition from industry clients, a managed development process, an iterative software development approach with self-created milestone management, using software tools that are prevalent in the software industry, learning from industry experts, formal product presentations, emphasis on problem solving and learning how to learn, and, exit exams that assess students' problem solving ability in addressing complex, industry problems [25].

A. The Software Studio

The Software Teaching Project has a section entitled 'What is a studio?' The following definition of a studio is a summary of that section, "A studio is a learning community of students and teachers, interacting in a creative, reflective process to develop some kind of product ,in a physical environment/space that enables collaboration." [8]. We use this definition as our basis, because of its provenance, as being developed by a number of studio-focused faculties from several universities in a large project over a number of years.

The studio concept for education was originally based on Schön's principles concerning the education of the practitioner [34]. In all professions, practitioners must decide which are the best techniques or processes to solve a particular problem. The software studio approach has been applied successfully to software engineering education, primarily in the USA and Australia.

The Carnegie Mellon University runs a Master of Software Engineering (MSE) program. The key feature is the MSE Studio, which is built around a year-long project for paying clients, providing students with a professional software development setting in which they can develop their skills [17] [33][38]. The design of this studio course was directly inspired by Schön's Reflective Practitioner concept [38].

Washington State University has also implemented a studio-based course for a computer sciences undergraduate course. The development of an adapted studio-based instruction for an introductory programming course called Pedagogical Code Review (PCR), which is modeled on the code inspection process used in the software industry. Students' self-efficacy (a measure of students' perceptions of their ability to program) was reported to decrease significantly among students who did not participate in the PCR while peer learning was reported to increase in the studio-based course [15][21][22]. Studio-based learning has promoted significant gains in students' intrinsic motivation, extrinsic motivation, self-efficacy, peer learning, self-regulation, and sense of community [22].

Auburn University in Alabama has also integrated studio-based learning into the undergraduate program of its Computer Science curriculum, starting in 2007. Unlike other universities' implementation of the studio approach, the activities of a studio-based at Auburn occur both in- and outside of class [22].

In Australia, the University of Queensland started a studio-based education by recognising several similarities between computer science and design sciences such as architecture. Based on these similarities, the Bachelor of Information Environment degree was developed by merging the three separate streams of computer science, interaction design, and project development into a studio-based curriculum. The subjects taught in all three streams converged in a mandatory studio-based course. The studio courses revolved around the collaborative creation and presentation of projects that relate to what is being taught in each stream. The feedback from the first intake of students in the studio-based curriculum was very positive [18].

Built upon the concept of studio-based learning, a redesign of the Bachelor of Information Management and Systems (BIMS) was implemented at Monash University, Australia. Lynch et al. [26] realised the need to better prepare students for jobs in industry. They proposed that studio-based learning (SBL) needs to be supported on three levels. First, the teaching space (physical layout of the classroom) needs to support collaboration amongst students. Second, the course needs to be taught using studio-based learning techniques. Lastly, coursework needs to be supported through a collaboration-oriented IT infrastructure [15]. The studio-based learning was implemented through a mandatory studio course. Again based upon the principle of Schön's reflective practitioner, the focus of the studio class was the groups of students to generate portfolios of past work and

achievements. These portfolios were then presented to faculty and the general public. The students responded positively to studio-based learning as they enjoyed interacting and collaborating with other students through working in groups. The students recognised that a studio-based learning further developed their knowledge and skill [26]. The studio facilitates learners' construction of knowledge by providing an environment in which the students can think, create and integrate [13]. Additional benefits of studio-based learning include requiring the students take some of the responsibility for their own learning process and experience [40]. Studio-based learning also provides a more comprehensive understanding of what the students may encounter once they are employed in industry [14]. The Monash program is often cited and has extensive educational investigations into the early years of running the course. The Monash studio is still running after 14 years, in a modified form.

The Software Development Studio is used in both the undergraduate and Master's SE courses for two semesters in a Polish university [24]. The Studio here is treated as an informal organisation with its own culture and processes and a number of development projects. This helps students understand how a real software organisation operates and they get to experience the different development roles necessary to complete a software project. The authors identified one of the challenges with this approach is related to balancing the assessment of a student project between the quality of its processes and its outcome [24]. Other issues they raise include scope and budget, customer involvement, organisational processes and policies, tutors/mentors assistance, motivation and quality of students, and infrastructure.

B. The Software Factory

The Software factory concept, a studio-like idea, originated in the University of Helsinki [12]. The proposed benefit of applying the software factory concept to software development education is to create practical learning of project activities following quality standards. In the software factory, students follow a defined and controlled process, which makes it easier for the students to understand the activities to be done. It also exposes the students to many problems which are common in enterprise, which assists an interaction among the students and a discussion with the teacher for the best alternatives for a solution. The software factory has been applied to software engineering education in Brazil, Finland and USA.

At the University of Helsinki, Finland, their Software Factory is considered a strategic investment in a new infrastructure supporting software engineering research, education and entrepreneurship globally [12]. The Software Factory here has the slogan: "Learn. Share. Grow", which implies a level of collaborative learning amongst the students.

In Brazil, the educational factory was designed taking the context of the course "Software Engineering Laboratory II" offered to senior undergraduate students (5th year) of Computer Engineering, at Escola Politécnica of the University of São Paulo (USP). Using a software factory in software engineering education allows the simulation of a real software development environment, in which undergraduate students execute the development activities and management activities to monitor and control the projects [35].

In the State University, Ohio, in the USA, the software factory is also applied to service-learning in a software engineering classroom, using Agile methods and collaboration with a technical communication course. Here, the students are exposed to the pressures present in the business environment, but at the same time they are assiduously educated in modern software technologies, development methods and testing techniques. After completing this course, most of the students valued the service-learning experience and the real-world skills they gained from their own involvement [16].

Both the software factory and software studio have the common aim of preparing students for jobs in the software engineering industry through the development of the necessary professional skills. These skills are technical, problem solving and collaboration. The difference between the software factory and the software studio is the former's emphasis on developing software engineering skills required by industry in a software development organisation. However, it does not appear to necessarily incorporate, and certainly not accentuate, the principle of reflective practice, which is a core approach in any studio approach

III. ISSUES OF IMMERSION AND REFLECTION

Software studio/laboratory/factory courses that have been implemented in undergraduate courses *form only part of the program* over one or two semesters, or are the mainstay of an entire postgraduate course. Only one, at Carnegie Mellon, is for the full 16-month duration of the course, and it is a Masters in Software Engineering course, with students who all have considerable prior industry experience [17]. One undergraduate course includes the studio approach for four semesters (rather than two semesters), but the college has very small class numbers, less than 10 students [29]. At Monash university, the software studio was implemented only for two undergraduate courses, Bachelor of Information Management Systems (BIMS), (one semester in the studio) and for the final year capstone project (two semesters), in the Bachelor of Software Engineering. BIMS was originally designed for students to have a studio subject throughout the course [26], but because of resource challenges currently only runs in the final year of the course (*personal communication with current course director*).

Those degrees which provide only two to four courses seem to lack exposure, and, the possibility for students to immerse themselves in the studio experience. This exposure does not provide necessary depth of experience to the students in the following areas:

- Experience in the various responsibilities of different developer roles
- Implications of decisions on long-term development
- Opportunities to learn from previous mistakes by reflecting on their work on a project and improving their practice on the next one.

The issue of reflection is fundamental to any studio which purports to be connected to Schon's reflective practice. It is the experience of the authors that reflection, with either undergraduate or postgraduate students, takes a significant time to learn, and, requires immersion.

In order to improve the way we teach SE/ICT at UTS, we propose an integration of the software factory and software studio educational approaches. The basis of the software factory approach is students working in a software development environment using professional tools and processes, with industry involvement in the course design and delivery. Studio-based learning builds upon the call to better prepare students for jobs in industry, by having the students working in a studio environment, in which reflection is a dominant theme [15]. The studio is seen as a realisation of software development as design practice.

The rest of the paper details our proposal and some of the insights we have gained so far during the feasibility, design and prototyping of our software studio.

IV. Proposed Solution: The Software Development Studio

We will move a substantial portion of the current undergraduate teaching into the Software Development Studio (SDS), wherein students engage on problems posed by industry, using industrial tools and working in development roles appropriate to their stages of learning and experience. Lectures, tutorials and laboratories would be reduced and significant course time spent in a studio environment. Practical development experience in the software studio will be combined with theoretical education and scholarship. Educationally, it will be based on reflective practice and developing software as design practice.

Another characteristic of the approach is that students will participate in the SDS throughout the duration of their degree course. This 'whole of degree' approach is significant because "effective education seems to be encouraged by both long-term engagement with the learning tasks and opportunities to exercise responsible choice in the content and method of study" [27] citing [30]. Students will experience different development roles and levels on a number of different projects throughout their degree program.

Industry involvement will be at various levels. The software development studio advisory board will comprise representatives from each of the companies supporting this program. Companies will provide development projects, and thus will benefit from the student work in having projects completed that may not be otherwise be tackled. Integrated software development environments with configuration management, version control and testing may also be provided by the industry. They will have direct input into the curricula, for subjects and the course as a whole.

As it is now considered a mainstream approach to developing large-scale, complex software products, an Agile software development methodology will underpin the work done on the projects in the software company. The decision of which software development methodology to utilise needed to be made early on as it impacts other design decisions, such as the physical teaching and learning environment (the classroom/studio layout), discussed briefly below.

The roles of the academics will be markedly different to their traditional teaching roles of lecturer or tutor, as their primary functions will be mentoring, coaching, guiding and facilitating in the master/apprenticeship studio model.

Similar schemes to the SDS exist, but no identical scheme has been implemented, as far as we can ascertain. A comprehensive program was designed in which students were required to take an eight-semester sequence, which put the students' newly acquired skills to work in a real software organisation staffed and managed by the students enrolled in the program [39]. Unfortunately, this course was not actually implemented, at least there seems to be no further literature on it, although the concept and the design were the inspiration for the 'software factory' courses in the USA and Brazil described earlier.

Whilst we obviously want to build on the experiences and recommendations of the software factory and software studio programs discussed in the previous section, there will be considerable differences in the way that a software studio program is offered at our university. The SDS will be specifically for undergraduate students, providing them with opportunities to learn in an environment that more closely mirrors professional practice. The experience will not be offered as a studio subject or unit, but across subjects throughout the degree program.

Initially, we are proposing to implement the SDS only for software education, i.e. for students who wish to become professional software developers or software engineers. It will start in full in Spring 2014, after extensive exploration, design and prototyping, as discussed below.

A. The Physical SDS Space

We have been allocated a dedicated studio room in our faculty's new building, to which we will relocate in early to mid-2014. The SDS has been designed to seat 30 students at one time, in groups of 5 to 6 students, each group in one of five hubs. Each hub has extensive whiteboards, on the walls and the partitions, a large monitor up on the wall, and in-desk power and data connections. The studio space also has a stand-up meeting area, with a waist-high table, an electronic whiteboard, audio-visual system and power and data connections. Two large sofas will be placed in the studio for comfortable, informal seating.

The design of the studio has incorporated aspects of a typical Agile, pair programming set-up with those of a traditional design studio, and more general collaborative teaching and learning spaces. We have also attempted to allow for both 'separate and together' ways of working, so that students can work alone without being disturbed, or work in teams on a design or development task.

B. Feasibility Study

A six-month feasibility study early in the project showed enthusiastic acceptance and support, within the faculty and university, for the proposed model of software engineering education and that it is logistically, technically and pedagogically achievable. This study also raised a number of issues that needed to be addressed, where possible in the prototyping stage, but some will be more effectively dealt with in the implementation of the full program.

1) Pedagogical Issues

The most pressing concerns that emerged from this study fall into this category.

a) Types and length of Projects

Students should participate in numerous projects over their degree [27]. This would provide opportunities for each student to work with a number of different teams and team leaders, on a spectrum of software products and software development methods.

b) Matching the motivation of students

Including students of differing capacities, differing work ethics and differing ambitions is a key factor in the design of the program. Some academics in our faculty suggested that we restrict the studio approach to the attainment of higher grades, at least for first and second year students – students wanting a higher grade (High Distinction or Distinction) must participate in the studio system, but the others would have the option to do the conventional lecture-tutorial-laboratory subjects only. This would also give a phased implementation strategy, which is a lower-risk strategy than starting out on a large scale with increased risks of failure.

c) Mentoring

A 'students mentoring other students' approach is risky, primarily because of selecting and training students, the attitude of other students to mentors [23]. Student mentoring is an aspect that will be explored once the studio is running. Academics as mentors raise workload and training issues for academic staff who will move into a mentoring role, as the capacity of the teaching staff will need to be developed in this area – most academics are used to a more conventional lecturing approach and will need re-skilling.

d) Assessment

An entirely different management of marking and grading of student work and performance will be required, in contrast to the more traditional, well-defined assessment tasks and marking. Bareiss [3] uses four viewpoints: team marks, individual work, peer review and the mentor's input. At this stage we are planning to have three broad categories of assessment criteria: product, process and reflection and students must perform satisfactorily in two of those three categories. This would help to mitigate the risks of students having to work with other students who may be less motivated or capable, as well as giving a more authentic assessment of the student's work over the time in the studio. This needs to be understood more deeply, but clearly will have to be based upon multiple viewpoints.

2) Other Considerations

The studio approach should be seen as supplementing lectures and tutorials, not necessarily replacing them. Other approaches need to be considered, before a final decision is made about the specific teaching and learning approach, as to whether it is a studio, a laboratory or some other mode.

We have done an analysis of the activities that students are likely to be doing at various stages of the project development process, based on an Agile methodology. We used these as input for design decisions with regard to the physical teaching and learning space, and will also use them for specifications on what will be required of the academic and industry mentors at these various times in the development process.

Finally, while real projects should be developed for real clients in the studio, students should not be treated like full-time software developers producing a commercial product from a financial perspective. In other words, the software studio should not be self-funded, reliant on income generated from the students' work, as this is not conducive to learning [32] or scholarship, so sponsorship for the everyday running of the studio is crucial.

C. Software Development Studio Prototype

The aim of the prototyping stage was to explore the studio-based approach and the issues raised above, and to manage risks, prior to introducing it as a formal part of the software development undergraduate courses. This prototyping stage moved on from the feasibility study outcomes to understand the pedagogical implications of introducing the approach. The prototype objectives include:

- Explore and evaluate different implementations of the approach;
- Ensure the studio ideas are well-explored and understood from educational and resource perspectives;
- Identify strengths, weaknesses, threats and opportunities; these may include: types and length of projects, differing capacities and motivations of students, mentoring, assessment, graduate attributes, reflective practice, management of perceptions and fears;
- Maximise effective risk management;
- Enable course objectives, although this is prototyping phase; and

- Implement a prototype which is practice-based, industry and faculty co-operatively driven.

1) Prototype I

In the first semester 2013, the first author sat in an Games Design Studio subject primarily in the role of ethnographer, making weekly field notes while observing the students' ways of working on a prescribed games development project using a Scrum process during their timetabled class time. She had no direct role in the classroom TAL activities on games development, nor in the assessment of any student work. She did have input into the design of the students' reflective journals and teaching them the process for completing these over the semester. The following are the overall conclusions; space prevents providing detailed ethnographic evidence.

a) Attitudinal Change

There was strong evidence of attitudinal change over the semester in most of the students, which demonstrated achievement of the 'harder' graduate attributes, attainment of which is one of the main aims of the studio approach (see V, below). For example, the students were initially resistant to 'process', finding it challenging to adhere to the disciplines of the Scrum process, but as the semester progressed, most of them learnt that it allowed them to make progress, by being able to methodically plan, and re-plan, their tasks and work allocation, in short cycles. So, it went from being a burden to being helpful.

b) Communication

There were also changes in communication. Later in the semester, groups noticed that they were more efficient when working together face to face more often (not just in class), and they progressed from an ad hoc approach to design and development to using co-ordination tools and documented communications.

c) Reflection

Reflection was still a "nuisance" for many students by the end of semester. Few students reached a level of reflection, rather than simply reporting, on their work, despite deliberate scaffolding throughout the semester and a significant percentage (20%) of their final mark being based on this task.

d) Sharing

Another unexpected issue was that many students continued to be reluctant to show, or submit, work in progress, putting off submitting the due task at the very last minute.

2) Prototype II

In the second semester 2013, we ran a software studio prototype, to understand how students and mentors react to the studio approach and to explore the enabling and limiting factors. We did this in a 2nd-year core subject, in which students work in assigned groups of 10, each with a project tutor/manager, to design and develop a complete software system from scratch. The subject is regarded, as being very challenging, by both students and staff, as it the students' first full system development experience. For the studio prototype, we asked for a volunteer group in the subject – the decision to participate needed to be unanimous within the group, given that they would have a different experience in the subject to the other students. Two groups enthusiastically volunteered to participate, and so we had 20 students, 2 academic tutors – the first author was one of these – and an industry mentor. We also had an ethnographer present for every studio session, observing and making field notes. Although the approach was different for these two groups, they developed their systems from the same set of requirements as the rest of the students in the subject and underwent exactly the same assessment.

The preliminary results from the fieldwork offer a number of insights, summarised below. Again, space prevents providing the evidence from the ethnography.

a) Mentors

The combination of academic and industry mentors was greatly valued by the students. While the academic tutors/mentors provided a structured learning process, guidance and direction on expectations and assessments, reflection and other subject organisation issues, the industry mentor provided advice from an up-to-date, "real-world" (students' phrase) perspective that the students specifically appreciated. Having both types of mentors also raised the issue of appropriate training and development for both academic tutors and industry mentors.

b) Process

As in Prototype I, the use of process was a significant hurdle, but to an even greater extent. 'Learning by doing' and having to cope with the technical challenges continually confronting them resulted in progressive development of students' confidence both in their understanding and use of the process, as well as effective collaboration within their team and the production of the system.

c) 'Doing time'

At the end of the semester, both groups of students stated that they would have preferred to have had more timetabled sessions together in the studio, as they gradually learnt that working together face-to-face regularly and for extended periods was far more effective and productive than trying to get tasks done individually in their own time and then integrating these in their class time.

d) Relationships

The third, and perhaps the most interesting, issue is that of relationships. The relationships between the students and their mentors set the tone for the type of learning community that developed over the semester. The students stated that they learnt different things and were supported in different ways from the three mentors, who had various approaches to mentoring and relating to the students.

The second type of relationship was the peer relationship. Intra-group relations were obviously important, particularly with

respect to how tasks were identified and allocated to the group members in each development sprint.

What was somewhat unexpected was the significance of the inter-group relationships. Having two groups or teams working together each week on their own projects allowed us to explore developments such as the change from competitive interactions to collaborative and supportive interactions later in the semester in which the two teams learnt from each other. This was the type of experience that set the two studio prototype groups apart from the rest of the students in the subject, who only met with their own team and tutor each week, focusing on what they were doing to achieve the project outcomes.

e) Innovation

The insights from the prototyping stage also relate to innovation, and what enables innovation and creativity, in the studio environment. We plan to explore this further in the last months of the prototyping phase.

V. Benefits of the Approach

Due to the cost involved, involvement and investment in this program must have benefits for the university, faculty, students, graduates, employers and industry partners.

The program can be seen as similar to an external internship program, with input into the type of training and experience companies usually give in an orientation/internship period for employed graduates, but it is deliberately integrated with tertiary course learning. Potential employers of students from the SDS will have a much clearer picture of what a student knows and can do when they have completed the course, particularly in relation to what will be expected of them in the professional work environment. The company will also need to spend much less time and money on company orientation/internship before the employed graduate is productive.

The emphasis of this approach is on gaining technical expertise and teamwork and interpersonal skills – in the context of software development and software engineering knowledge, in contrast to management or sales skills, for example – through the process of producing commercial software products as part of a development team, whether this be a project requested by an industry partner, or an innovation developed by students as part of an innovative entrepreneurial process. The attainment of graduate attributes has become a structuring goal in the higher education sector in Australia. A software studio enables the development and assessment of these more holistic skills and, in particular, those attributes that may be considered as employability skills [10] in the context of the discipline and professional practice.

Industry will get graduates who are experienced across the whole life cycle of development. Even if it is not with the company's specific tools and methodologies, the graduates would have experience in the typical activities that would be part of a company's internship period. It is an aim that students will be productive, and deeply rooted in practice, in the company much sooner after graduation. Companies will also be taking on graduates who can be better assured to be independent learners, reflective about their practice and not simply academically competent in the area of software development.

As students will have experience in a variety of roles within the development company, both graduates and their employers will have a much clearer idea of what roles suit them best and in which they are likely to be most productive and of benefit to the company and to the students' own career paths.

VI. Concluding Remarks

We propose a software learning and development approach for undergraduates that has a strong industry-collaborative character, emphasises reflective design practice, and incorporates the success factors of the software factory and software studio approaches. The major innovations in this proposal are:

- a software studio is being established, with which the students will be closely associated throughout their degree;
- it is aimed specifically at educating software developers and software engineers, rather than as part of a more general ICT degree;
- it will be part of undergraduate degree courses, with subjects 'contributing' credit points to fund the studio, in return for satisfaction of subject objectives;
- there will be significant industry involvement in the studio, for instance, as members of the studio advisory board and in providing projects;
- students will work in various roles in different teams on several industry suggested projects;
- students work in the studio for the duration of their course, not just in their final project or capstone semesters.

After the completion of the prototyping stage for the software studio, in which we will also identify evaluation criteria for its successful implementation, we look forward to implementing this approach in full in our school from mid-2014.

Acknowledgments

This work was made possible by a Faculty Teaching and Learning Grant 2012/2013. We acknowledge the contribution of our colleague Jocelyn Armarego from Murdoch University. We also acknowledge the staff members, mentor and students who participated in the prototype at UTS.

References

[1] Australian Computer Society (ACS). Australian Computer Society Statistical Compendium 2012 (Annual Report). http://www.acs.org.au/news-and-media/news-and-media-releases/2012/acs-statistical-compendium-2012. Retrieved on 30 October 2013.

[2] Armarego, J. & Fowler, L. Orienting Students to Studio Learning', Proceeding of the 2005 ASEE/AAEE 4th Global Colloquium on Engineering Education. Australasian Association for Engineering Education, 2005.

[3] Bareiss, R. & Griss, M., A Story-Centered, Learn-by-Doing Approach to Software Engineering Education, SIGCSE'08, ACM Press, 2008.

[4] Koppi, T, & Naghdy, F. Discipline-Based Initiative: managing educational change in the ICT discipline at the tertiary education level, Australian Learning and Teaching Council (ALTC) Final report. Wollongong: University of Wollongong, 2009.

[5] Koppi, T., Ogunbona, P., Armarego, J., Bailes, P., Hyland, P., McGill, T., Naghdy, F., Naghdy, G., Pilgrim, C. and Roberts, M., Addressing ICT curriculum recommendations from surveys of academics, workplace graduates and employers, OLT Final Report. Wollongong: University of Wollongong, 2013.

[6] Hazzan, O. The reflective practitioner perspective in software engineering education, The Journal of Systems and Software, 63, 2002.

[7] Studio Teaching Project, 2008-2009, active in 2013. http://www.studioteaching.org/, retrieved 13December 2013.

[8] What is a studio?, Studio Teaching Project, 2008-2009, active in 2013. http://www.studioteaching.org/index.php?page=what_is_studio

[9] Narayanan, N H, Hundhausen, C, Hendrix, D, & Crosby, M. Transforming the CS classroom with studio based learning. SIGCSE'12, Raleigh (NC), 2012.

[10] DEEWR. Employability Skills Framework Stage 1 – Final Report, 2012. Canberra: Australian Government. DEEWR Retrieved on 1September 2013 from http://foi.deewr.gov.au/system/files/doc/other/employability_skills_framework_stage_1_final_report.pdf

[11] Kuhn, S.. Learning from the architecture Studio: implications for project-based pedagogy. International Journal of Engineering Education, 17(4 and 5), 349-352, 2001.

[12] P. Abrahamsson, P. Kettunen, and F. Fagerholm, "The set-up of a software engineering research infrastructure of the 2010s". in Proceedings of the 11th International Conference on Product Focused Software (PROFES '10), pp.112-114, 2010.

[13] A. Carbone and J. Sheard, "A studio-based teaching and learning model in IT: what do first year students think?", Proceedings of the 7th annual conference on Innovation and technology in computer science education, 2002, pp. 213-217.

[14] A. Carbone, K. Lynch, A. Barnden and C. Gonsalvez, "Students' reactions to a studio-based teaching and learning philosophy in a three year IT degree", Proceedings of the Annual International Conference of the Higher Education Research and Development Society of Australasia (HERDSA), 2002.

[15] A. Carter and C. Hundhausen, "A review of studio-based learning in computer science", Journal of Circuits, Systems, and Computer (JCSC 27), October, 2011.

[16] J. Chao and J. Brown, "Empowering Students and the Community through Agile Software Development Service-Learning", P. Abrahamsson, M. Marchesi, and F. Maurer (Eds.): XP 2009, LNBIP 31, pp. 104–113.

[17] A. Damasceno, "MSE studio project: the viewpoint of a UC student", CSEE&T 2011, Waikiki, Honolulu, HI, USA, 2011.

[18] M. Docherty, P. Sutton, M. Brereton and S. Kaplan, "An innovative design and studio-based CS degree", SIGCSE 2001, 2/01 Charlotte, NC USA, 2001.

[19] D. Garlan, D. P. Gluch and J. E. Tomayko, "Agents of change: educating software engineering leaders", IEEE, November, 1997.

[20] K. Garg and V. Varma, "People issues relating to software engineering education and training in India", ISEC'08, Hyderabad, India, 2008.

[21] C.Hundhausen, A. Agrawal, D. Fairbrother and M. Trevisan, "Integrating pedagogical code reviews into a CS 1 course: an empirical study", SIGCSE'09, Chattanooga, Tennessee, USA, 2009.

[22] C. Hundhausen, A. Agrawal, D. Fairbrother and M. Trevisan, "Does studio-based instruction work in CS 1? An empirical comparison with a traditional approach", SIGCSE '10, Milwaukee, Wisconsin, USA, 2010.

[23] Kinniburgh , J., "A Culture of Success: Building Depth into Institution Wide Approaches to First Year Transition", 16th International First Year in Higher Education Conference (FYHE 2013), 2013

[24] Kopczynska, S.; Nawrocki, J.; Ochodek, M..,"Software development studio — Bringing industrial environment to a classroom," Software Engineering Education based on Real-World Experiences (EduRex), First International Workshop on , pp.13-16, 2012.

[25] J. Liu, J. Marsaglia and D. Olson, "Teaching software engineering to make students ready for the real world", the Consortium for Computing Sciences in Colleges, 2002.

[26] K. Lynch, A. Carbone, D. Arnott and P. Jamieson, "A studio-based approach to teaching information technology", Proceedings of the Seventh world conference on computers in education: Australian topics, Vol. 8, 2002, pp. 75-79.

[27] J. Leaney, C. Peterson, C. Drane, "Computer systems engineering in large groups," Proceedings of Frontiers in Education Conference, FIE '96, vol.3, pp. 1491-1494, 1996.

[28] N. R. Mead, "Software engineering education: How far we've come and how far we have to go", The Journal of Systems and Software, 2008, Vol. 82. pp. 571-575.

[29] T. Nurkkala and S. Brandle, "Software studio: teaching professional software engineering", in Proceedings of the 42nd ACM technical symposium on Computer science education (SIGCSE '11), 2011.

[30] Ramsden, P. "Learning to Teach in Higher Education", Routledge, London, 1992.

[31] Ramakrishnan, "MUSE studio lab and innovative software engineering capstone project experience", ITiCSE'03 June 30-July 2, 2003.

[32] D. Root., R. Mel and G. Taran, "Exporting studio: critical issues to successfully adopt the software studio concept", 21st Conference on Software Engineering Education and Training, 2008.

[33] Shaw, M., Herbsleb J., Ozkaya, I. and Root, D., "Deciding what to design: closing a gap in software engineering education", P. Inverardi and M. Jazayeri (Eds.): ICSE 2005 Education Track, LNCS 4309, 2006. pp. 28-58.

[34] D. A. Schön, The Reflective Practitioner, Basic Books, Inc., New York, 1983.

[35] F. Siqueira, G. Barbarán and J. Becerra, "A software factory for education in software engineering", 21st Conference on Software Engineering Education and Training, IEEE Computer Society, 2008.

[36] E. Stiller and C. LeBlanc, "Effective software engineering pedagogy", CCSC: Northeastern Conference, 2002.

[37] Tan, G., & Venables, A. "Survival mode: The stresses and strains of computing programs review". Journal of Information Technology Education, vol 7, pp.33-43, 2008.

[38] J. E. Tomayko, "Carnegie Mellon's software development studio: a five year retrospective", IEEE, 1996, pp. 119-129.

[39] J. Tvedt, R. Tesoriero and K. Gary, "The Software Factory: combining undergraduate computer science and software engineering education," Software Engineering, 2001. ICSE 2001. Proceedings of the 23rd International Conference on , pp.633-642, 2001.

[40] L.Williams, L. Layman, K. M. Slaten, S. B. Berenson and C. Seaman, "On the impact of a collaborative pedagogy on African American millennial students in software engineering", 29th International Conference on Software Engineering (ICSE'07), 2007.