Masters-level software engineering education and the enriched student context

Conference or Workshop Item

Masters-level Software Engineering education and the enriched student context

Jon G. Hall*, Lucia Rapanotti*
*Department of Computing and Communications
The Open University, Milton Keynes, MK7 6AA, UK
Email: Jon.Hall@open.ac.uk; Lucia.Rapanotti@open.ac.uk

***Abstract*—Currently, adult higher education software engineering pedagogy isolates the student in a controlled environment during delivery, with application of their learning temporally distant from their professional practice. Delivering software engineering teaching that is immediately relevant to professional practice remains an open challenge. In this paper, we discuss a new pedagogical model which addresses this problem by embedding the validation of the student's learning within their rich professional context. We discuss our experience of applying the model to the design and delivery of a new post-graduate software development module, a core component in our new software engineering Masters qualification at the Open University, UK, a market leader in adult higher education at a distance.**

I. Introduction

A major challenge of software engineering education is marrying theory and practice, so that the learner develops both a deep understanding of the academic discipline and the skills to solve real-world software engineering problems, in all their facets and complexity. In the last decade, value has been found in pedagogical approaches which try and meet this challenge by presenting learners with real-world open-ended situations, rather than toy examples and fictitious software problems: learning through tackling real-world open-ended problems narrows the gap between what students do as part of their study and what they encounter when working in the industry and, as such, is a welcome advancement in the way our discipline is taught. This is, for instance, the case in problem-based learning, in which groups of learners presented with real-world open-ended situations learn through a learner-centered process of identifying problems, conducting the necessary research and proposing appropriate solutions. In this approach, the academic takes the role of a facilitator, while students take the initiative and greater responsibility for their learning. Applications of problem-based learning to software engineering education are increasing, with examples reported in the literature both for traditional class-based face-to-face education [1], [2], and for distance education mediated by online communication [3].

Yet an element of artificiality remains: the exercise only simulates the real-world situation in the confines of the classroom (physical or virtual), and some have questioned the extent to which this actually prepares engineering students for real-world practice [4], [5]. Other software engineering courses add value through work placement instead, often coupled to capstone projects [6]. Studies on the effectiveness of this approach are lacking, but anecdotal evidence seems to indicate that there remain difficulties linked to the degree of alignment between work experience and what is taught in the classroom, the availability of suitable hosting organisations, and the variability in the quality of the work experience, in particular in relation to the nature of the work students are actually allowed to participate in.

In this paper we discuss a pedagogical model, developed at the UK's Open University (OU), which removes the temporal distance from academic instruction to real-world experience by blending the learner's study and their rich real-world context. Initially aimed at our part-time research students [7], we have developed and refined the model over the past decade for MPhil and PhD students. First applied in 2013 to the post-graduate-level teaching of Information Security, we report on its application to the design of a post-graduate software engineering module, offered for the first time in May 2014. This paper focuses on this latter application and reports both on the pedagogical model, its impact on the module design and the outcome of an initial partial evaluation we have conducted. The initial data are encouraging: students embrace the new model, deriving value from it both for themselves and for their organisation.

II. Background

A. The university and programme of study

The UK Open University (OU) is a world leader in part-time adult higher education at a distance. Currently, over 73% of our 200,000 students work full or part-time during their studies, with higher percentages for students enrolled on our post-graduate programme in Computing and ICT. Specifically, of the 100 students who started our module in May 2014, 89% declared to be in full-time employment and 5% in part-time work.

While we have taught aspects of software engineering in modules both at under- and post-graduate levels for over a decade, a separate Masters level qualification in software engineering was introduced only in May 2014. At its core are two modules that span the body of knowledge [8]; the first (the subject of this paper) focuses on a range of software engineering theories, principles and techniques across the life cycle, with particular emphasis on for problem definition, analysis, design, implementation and testing.

B. The OU distance education model

OU students conduct their part-time study entirely at a distance. They are assigned to small tutor groups (up to 17 students at post-graduate level) with all interaction with tutors and peers via communications technology. Study is enabled by bespoke pedagogical materials delivered to each student, driven by a shared study calendar and validated at many assessment points. Increasingly, a Virtual Learning Environment (VLE) provides the single point of access to a module's resources, services and activities; this includes the module in question.

C. Student body and entry skills

The 'openness' of the OU means that most of our courses have no formal prerequisites; we give credit to the practical and professional skills that a student may have. Our new software engineering qualification attracts students at many levels of software development knowledge, both formal and practical, a situation compounded both by the 'open entry' nature of the IT profession and the spread of technologies used in the market place. Many practicing professionals are self-taught, have retrained from other professions, or are highly specialised, with developed skills and knowledge gaps that are highly dependent on their journey into IT.

Our students use our post-graduate offering to gain academic recognition for their practical skills, to develop new skills and to fill knowledge gaps that prevent their professional progression. Most students have work and family responsibilities and want flexible, relevant education which delivers value both to them and to the organisations they work for. Indeed some organisations sponsor their employees' studies.

D. Learning outcomes and assessment

In contributing to a post-graduate qualification, the module delivers a wide range of learning outcomes organised around:

Knowledge and understanding: including a wide range of software engineering theories, principles, techniques, practices, standards and systems, and their application personally and in business;

Analysis and Synthesis: including the ability to integrate diverse software engineering knowledge, techniques, tools and practices into a coherent whole, making appropriate abstractions and dealing systematically and creatively with complexity.

Transferrable skills: such as communicating effectively with technical and non-technical audiences; argumentation; critically reflection on and evaluating of their own work and the work of others.

Professional and research skills: including the consideration of risks, legal issues, cultural and ethical factors and business needs; being able to identify needs, articulate goals, locate and employ resources and follow action plans in support of independent learning and professional development.

The module uses three types of assessment to track student progress towards these learning outcomes: formative assessment, where students make critical choices which may affect their study success; summative continuous assessment, for tracking student progress; and summative end of module assessment, which focuses primarily on research and other professional skills.

III. Pedagogical design: blending theory and practice

A. Theoretical model

Quine, one of the most influential philosophers of the 20th century, observed [9]:

> Total science, mathematical and natural and human, is [...] underdetermined by experience. The edge of the system must be kept squared with experience; the rest [...] has as its objective the simplicity of the laws.

We have noted elsewhere that Quine's observation suggests that generative research in disciplines that span both theory and practice can fruitfully be conducted within a part-time student's rich professional context and we have developed a research supervision model that does so [7].

Quine's observation does not, however, say how to teach theory within a practical discipline. Issues include: are all parts of the theory relevant? If not, which are relevant and in which order should they be taught? How should theory be assessed from its practical application? What value is delivered if the student's rich context is used?

Much of software engineering has developed in practice, and only epistemologically post-rationalised. As such, its theories exhibit many of the issues mentioned above. Our thesis is that there is value added for both the practitioner/student and their organisation in allowing them to bring their rich professional context into a supportive taught software engineering module, with teaching mechanisms based on the experience of the application of theory in satisfying the student's needs within that rich context. This value arises from, in Quine's words, '[keeping] the edge of the [theory] squared with experience': specifically, through the application of their learning to an organisational problem, students deliver into their rich professional context valuable developmental artefacts ranging from a detailed problem understanding through to candidate solutions.

B. Module design

In traditional learning, the educator is wholly in control of the detailed teaching agenda, *i.e.*, the sequencing of teaching must respect only the theoretical relationships between its elements. In problem-based learning, constraints are added to ensure that specific problems are treated at specific times. Our *problem-oriented* model adds further constraints, in that the educator must organise the teaching to deliver fit-for-purpose conceptual tools *before the student needs them*, so as to enable the development of understanding of, and ability to, solve real-world problems in a wide range of student contexts.

Whereas it is not at all obvious how these additional constraints can be satisfied in the general educational setting, we observe that, in process-based disciplines[1], the processes that

[1]Which includes most of engineering, hence software engineering and information security, another module taught through our problem-oriented approach.

the student will use to understand and solve their problems are a valuable organising concept. On the module in question, for instance, in Block 1 students learn principles and techniques of early software lifecycle, from requirements and domain analysis to software specification, engaging practically through, for instance, the capture and validation of requirements within their professional context, and UML modelling with activity and class diagrams; in Block 2, the focus changes to software design, construction and testing, engaging with design practices, including the application of design principles and patterns, UML modelling with interaction and state diagrams, and software construction and testing in Java; finally in Block 3, they further develop skills acquired in the previous blocks to revisit their software solutions based on software architectures and frameworks, engaging in design and Java development which make use of standard software components, services and protocols.

C. Validation of learning

Disciplines with a process basis thrive on critical reflective practice. Our observation is that the teaching of process can thus form an effective basis for validation. For the software engineering module, for instance, theories, principles and techniques are taught for application by a student within their rich real-world context. The student in then required to reflect critically on their experience and to build a commentary on the extent to which the applied theories, principles and techniques have been fruitful, appropriate, deficient, over-complex, or just plain wrong, therein.

This theory-supported process application within the student's rich context, followed by guided critical reflection, is the main vehicle for assessment and validation of learning. In addition, formative assessment takes place early in their study to arrive at an appropriate choice of development problem and organisational context, this choice being a key success factor. They are expected to make steady progress with their chosen development throughout their study, assessed at the end of each block by their tutor, who also prepares feedback. Alongside their submitted software development artefacts, which demonstrate to which extent they have mastered specific approaches and techniques, they must also provide the written commentary we have just discussed. The guided reflection also includes the rationale for their choices in the process application within their context, and lessons learnt, and a critical reflection on the teaching against real-world practice.

How does the rich context model affect the tutor's work? Tutors — experienced software professionals — are chosen for their ability to deal with the variance in the student's context. Nevertheless, a generic marking scheme provides consistent marking across the student cohort, despite the diversity of development work to assess. This marries qualitative marking criteria related to the module's learning outcomes with quantitative measures which allow the tutor to assign a final grade, one of fail, pass, merit or distinction.

D. Additional value

Value may accrue for the organisation, too, under our model; in addressing an organisational problem, the student delivers solution artefacts, such as problem understanding, stakeholder requirements, even candidate software solutions. There is also additional value for the student whose profile can be raised within the organisation.

IV. Early evaluation

The module was presented for the first time in May 2014. One hundred students started the module, organised in six separate tutor groups. The retention rate was 84%, compared to the previous 5-year programme mean of 76.7%. Students' summative continuous assessment had mean 74% (s.d. 13.5%) and end of module assessment had mean 62.8% (s.d. 19.14%).

Each tutor had over 10 years experience of supporting OU students; for two tutors, however, this was their first experience at post-graduate level. All were experienced software professionals.

With one hundred students, the spread of organisations across sectors and the variety of development projects was remarkable: from large companies to SMEs; from sole traders to local authorities; from large financial institutions to cleaning companies; from a system to process prices and quantities of mutual funds in a financial institution, to an online foreign language learning system. As well as confirming the richness of our students' context, these data also provide detailed, valuable insights into our students' professional background, something which is not routinely accessible to an educator. With data available every presentation, it will be a rich source of information as to how the profession changes over time.

Our initial analysis of a random sample of continuous assessment scripts ($n = 40$, $N = 300$) has provided some insight as to the value delivered to the student, in terms of their professional development. Many students acknowledged the utility of what we teach, even starting from a skeptical position. Comments include:

> *On first glance the approach to building the requirements document, use case, activity diagrams, class diagrams and [...] operational specification appear to be a bit tedious for the size of [my] problem [...] However it soon became apparent how useful these tools are [...] when interacting with the stakeholders. I suspect the interactions with stakeholders wouldn't have been as effective without the models.*

and

> *[...] the most useful artefacts in terms of the analysis process were the textual descriptions of use cases and the glossary [as they] provide a means of sharing a precise understanding of the problem domain with stakeholders.*

and

> *While at first I was very sceptical of writing unit tests for relatively simple operations, I've found it very useful and I've noticed that it has enabled me track flaws in the design.*

However, it is also clear that what we, as academics, see as good software engineering does not necessarily square with

what is feasible or perceived useful in practice. For instance, one student remarked:

> *I am rather skeptical about [collaboration diagrams]. In a situation where a new application is designed from scratch, I can see this being used. In practice a developer inherits an undocumented thrown-together application on which she has to build a new feature before the end of the week. Taken together with the widespread use of Agile, I can't really see a developer following all these steps to arrive at an acceptable solution, although he should.*

While what we teach seems to align quite closely to the experience of most of our students, it does not cover practices in specific industries, notably embedded software engineering:

> *the technique of unit testing is something that I have not come across in my career as an embedded software engineer so far. While this technique is not designed to replace formal verification and could potentially increase the development time, I personally enjoyed the psychological benefit of seeing the green tick when a unit test had passed.*

One student went as far to perform his own small evaluation of some UML techniques and wrote

> *I had the occasion to try out UML diagrams on two non-technical colleagues. The response varied according to which type of diagram was discussed: Activity diagrams were understood intuitively. Use case diagrams proved less intuitive, as they were understood as sequences to be read top-to-bottom. [...] Class diagrams – to clarify authorization [sic] concepts (user, role, session, permission) – did not prove useful.*

Beside the continuous assessment scripts, we have also received some early feedback on the students' experience on the module as a whole via the module forum. The comments so far are very positive. One stated "*it was definitely useful for my career and my work*". Another student, an experienced software developer, noted that the module

> *reinforced knowledge that I had already gained through work. In some cases it helped me to identify sloppy practices that had crept into my approaches over the years [and] inspired me to investigate further some areas that I had never made the time to investigate before.*

Room for improvements were, of course, also noted. For instance, "*teamwork would be good to add in somehow*" or

> "it was useful to get a basic understanding of the [Java EE] architecture, but for the time of the course, it was impossible to give a good enough detailed knowledge to really gain much from the practical exercises"

Finally, from the tutor's perspective, initial concerns as to the potential high workload related to the new practice-based assessment quickly disappeared with the tutors acknowledging that the marking scheme provided was very efficient and that assessing original and varied project work was more enjoyable and rewarding than that of case studies.

V. Conclusion

The paper has discussed a pedagogical approach which attempts to marry academic teaching and professional practice in the context of a new distance learning post-graduate level software engineering module. While acknowledging the limitations and possible bias of our initial evaluation, there are some encouraging signs that the model is effective: students appear to have taken our teaching into their rich context and professional practice and found value both in terms of their professional development and in widening their knowledge and understanding of the discipline.

From an academic perspective, the model required a rethink of the way we design our teaching and its delivery in order to be timely, relevant and applicable across a wide range of student selected software development problems. The model also required the careful design of a standardised assessment strategy, to be both scalable and able to deliver post-graduate level validation. The situated problem-oriented summative continuous assessment provides a valuable feedback loop to help us evaluate and calibrate our teaching of software engineering theories and techniques against the stringent requirements of current professional practice.

The new model required a leap of faith: we had to combat a fear of losing control of a key component of the module in allowing the students to bring their choice of development problem into their learning rather than giving them standard problems to solve. Feedback from this first module presentation indicates that this aspect of the module has been valuable for students, tutors *and* educator.

Future work is being commissioned that will analyse more systematically the full set of data from the first presentation of the module to allow us to be more conclusive as to the qualities and limitations of our approach. We will also compare outcomes from this module with those on the related Information Security module which makes use of the same pedagogical approach.

Acknowledgment

To our students, for making us better teachers.

References

[1] I. Richardson, L. Reid, S. B. Seidman, B. Pattinson, and Y. Delaney, "Educating software engineers of the future: Software quality research through problem-based learning," in *Software Engineering Education and Training (CSEE&T), 2011 24th IEEE-CS Conference on*. IEEE, 2011, pp. 91–100.

[2] J. D. Delaney, G. Mitchell, and S. Delaney, "Software engineering meets problem-based learning," *The Engineers Journal*, vol. 57, no. 6, 2003.

[3] L. Brodie, H. Zhou, and A. Gibbons, "Steps in developing an advanced software engineering course using problem based learning," *engineering education*, vol. 3, no. 1, pp. 2–12, 2008.

[4] J. Perrenet, P. Bouhuijs, and J. Smits, "The suitability of problem-based learning for engineering education: theory and practice," *Teaching in higher education*, vol. 5, no. 3, pp. 345–358, 2000.

[5] J. E. Mills, D. F. Treagust *et al.*, "Engineering education—is problem-based or project-based learning the answer?" *Australasian Journal of Engineering Education*, vol. 3, pp. 2–16, 2003.

[6] L. Johns-Boast and S. Flint, "Simulating industry: An innovative software engineering capstone design course," in *Frontiers in Education Conference, 2013 IEEE*. IEEE, 2013, pp. 1782–1788.

[7] J. G. Hall and L. Rapanotti, "Enterprising research skills: academia's changing role," *International Journal of Learning and Intellectual Capital*, vol. 10, no. 1, pp. 1–17, 2013.

[8] P. Bourque and R. Failey, Eds., *Guide to the Software Engineering Body of Knowledge, Version 3.0*, 2014.

[9] W. V. O. Quine, "Two dogmas of empiricism," in *From a Logical Point of View*, 2nd ed. Harvard University Press, 1961, (Originally published in The Philosophical Review 60 (1951): 20-43) Text available from `http://www.ditext.com/quine/quine.html`.