Chapter 16

Informing eLearning software development processes with the student experience of learning

and Social Work

The opportunities provided by eLearning technologies to enhance the student experience are encouraging universities to systematically invest into new eLearning projects. The implications of this trend for ensuring the quality of the student learning experience are serious and complex. One important aspect is how educational software being developed and used is related to the quality and effectiveness of the students' learning, and for software engineers, how the software can be developed to produce the most productive experiences.

Software engineering is a comparatively young discipline in universities, but it has already had a significant influence on modern experiences of education. As with other engineering specialisations, the maturity of software engineering is recognised only when it has established recognisable methodologies supported by solid evidence. Software engineering, outside the educational domain, employs numerous, well supported and recognisable methodologies (Pressman, 2005). For the purposes of education, however, this evidence is not sufficiently illuminative as it does not concern itself with the needs of students in relation to learning outcomes, the way they approach the use of the technologies to support their learning or their conceptions of how learning is supported by technologies.

There are significant challenges for software engineering research focusing on learning technologies. While the success of a business software system can be measured in ways such as a reduction of costs, or time, or increase of business efficiency, in education such quantitative assessments are less useful. The benefits from education are often ephemeral, realised in the medium or long-term and hard to capture. To address these challenges, current software development methodologies for learning technologies try to break down the development problem into smaller parts, each with its own quantitative measure. This often does not work as the smaller parts do not necessarily cohere to create a meaningful whole. Consequently, there is considerable room for learning software development processes to be improved, especially if the processes can be shaped by the nature of educational experiences they are designed to support. Unfortunately, this is no easy matter as it requires the collaboration of teams of people from different disciplines, most notably software engineers and educational researchers. This chapter addresses the question of how software engineers and educational researchers can productively collaborate to improve software development processes for learning technologies so that the technologies are more likely to help students learn effectively.

Studies into associations amongst learners and their use of technology influencing its design is an important research focus in the field of learning. One area of research related to this focus is computer supported collaborative learning (CSCL). A particular focus of research into CSCL has investigated associations amongst software tools and learners (Lehtinen, Hakkarainen, Lipponen, Rahikainen & Muukkonen, 2002). Some studies have investigated students' use of mathematical software programs for collaborative learning (Derry, 1990; Reusser, 1996). Some have investigated

students' use of collaborative software aiding experimental research methodology and statistical inferences (Lehtinen, Hämäläinen & Mälkönen, 1998). Others have investigated probability inquiry and use graphical representations to encourage collaboration amongst learners. (Enyedy, Vahey & Gifford, 1997). The emphasis in these studies is more about how the technologies support the interaction, rather than how the student experience influences the design of the technologies.

The research discussed in this chapter distinguishes itself from CSCL research in its focus. Rather than investigating mainly how software technologies may support collaborative learning, it investigates how students experience technologies that are developed to support different learning experiences, and then uses that experience to inform the subsequent software development processes to improve their design and underlying intentions related to improving learning. This focus of the research is essential if we are to help educationalists and educational software engineers who are innovating with learning technologies to improve student experiences of learning.

To clarify how educational concerns can be practically and sustainably integrated into software development processes, we look at software engineering projects at the University of Sydney where we developed eLearning applications using evidence of the relationship they have to our students' learning experience. We use the outcomes of our own previous research studies (Carroll & Markauskaite, 2006; Ellis & Calvo, 2006; Ellis, Calvo, Levy & Tan, 2004; Turani, Calvo & Goodyear, 2005) to produce a variant of the spiral methodologies that is informed by the student experience. The model we use to describe students' experience of learning is next described. Then the *spiral-Ed* methodology we propose is described. This is followed by a description of the projects for which we have been using it. Finally, the teaching context and results are described.

Prior research

The research reported on in this chapter draws together ideas from the current literature in software engineering, particularly on the spiral model development methodologies (Boehm, 1988; Pressman, 2005) and on research into student learning in higher education (Biggs, 2003; Entwistle & Ramsden, 1983; Laurillard, 2002; Prosser & Trigwell, 1999).

In software engineering, several methodologies have been developed over the last three decades. Amongst these, the spiral models, studied since the 1980's, have two common features: an iterative approach for incrementally growing a system's degree of definition and implementation and a set of milestones for ensuring stakeholder commitment to feasible and mutually satisfactory system solutions (Boehm, 1988). The advantage of the spiral model for software engineering is its principle of committing resources incrementally to researching and defining the problem, and then developing the software, instead of a large commitment of resources before its prospects for success are understood.

Normally engineers decide what to build after analysing the requirements through 'understanding what the customer wants, analysing need, assessing feasibility, negotiating a reasonable solution, specifying the solution unambiguously, validating the specification, and managing the requirements as they are transformed into an operational system' (Pressman, 2005, p. 144; Thayer & Dorfman, 1997). The challenges of doing this analysis are well known, particularly on industrial and business systems. When eLearning systems are developed the analysis becomes more complicated. On one level, engineers look at what students and teachers will do with

the system, but at a second level, educational research has shown that the picture is much more complex, and a complete analysis must include other aspects of the learning experience and students' perceptions.

Principles of design informed by the student experience of learning

One of the challenges of developing software systems for learning is that software engineers do not yet have efficient ways of including the student experience of the technologies into their design. What we are proposing are a few simple questions which, if systematically used by software engineers, will go some way to providing input directly from students, into design processes.

In several research projects (Ellis et al., 2004; Ellis & Calvo, 2006) we have used a phenomenographic model of learning to investigate the quality of learning through the use of specific software. This model investigates the student learning experience by dividing it into key aspects: the *how* or its structural aspect, and the *what,* its referential aspect. The *what* is often referred to as the direct object of learning. The *how* aspect can be subdivided into two further parts: the act of learning and the indirect object of learning. Related to these key aspects of learning, we have also investigated student perceptions of aspects of their learning context (Ramsden, 2003).

In our studies, we have found that the quality of what students' think they are learning has been logically and positively related to the quality of their approach to using the technologies in their experiences of learning. This research has found that students, who tend to adopt more meaningful approaches to the use of learning technologies, often perform relatively better than other students who adopt poorer approaches to the use of learning technologies. Significantly, the quality of student experiences of learning using technologies has also been logically related to their perceptions. Higher quality experiences tend to be associated with positive perceptions of the learning context.

From these studies, we have adapted the key research questions more generally to apply to software development processes. We seek to identify associations amongst the learners' *conceptions* and *perceptions* of, and *approaches* to using the software, to the way the functionalities of the software are developed. In software engineering, these are unusual questions as student conceptions, perceptions and approaches are variables not normally included in the engineering processes simply because they are not applicable to non-educational systems.

A methodology for eLearning software development

Software development processes, influenced by the student learning experience, have been adopted by the Web Engineering Group at the University of Sydney. The group has developed a number of eLearning applications and activities which have been shaped by feedback from students using them for coursework. What we have learned from these projects is offered here as the spiral-Ed methodology which can be used by other developers. The purpose of this methodology is to allow those involved in developing software, (software engineers, instructional designers, interface designers and teachers), to be informed as to the particular student-related needs and goals of learning technology. The methodology proposed suggests that a software development team needs to incorporate these educationally orientated roles.

For example, let us consider a project in which a discussion tool is being developed. The engineer will come to the project with a number of skills and a tradition of

evaluating software systems that include a requirement analysis process. If the development team were evaluating the discussion tool, they would inquire as to the functional and non-functional requirements. These requirements describe what the students do with the system (for example, post or read a text document), and how the tool behaves (for example, how much time it takes the system to respond). For business software applications, it is often self-evident why people use the software the way they do, because their use is often motivated by a pragmatic goal, such as accessing some particular information or automating a task. In contrast, for learning technologies, the reason students do things, and how they do them, can often be more important than the actual completion of the task using the software. Table 1 shows examples of the 'metafunctional' dimension, or characteristics of the software which should improve the learnability of the student user experience.

Table 16.1: Example of functional and metafunctional requirements of discussion tools

Functional/Usability Requirements	Metafunctional/Learnability Characteristics
Time for posting	What have you learned through the discussions?
Time for assessing the postings	How do you approach using the discussion tool?
Number of postings	Why do you use the discussion tool in the way you do?
Organisation of fora	What aspects of your experience of using the discussion
Search functions within fora	tool prevented you from learning?

Traditional software development approaches would normally only consider the requirements listed in the first column in Table 16.1. In the approach argued for here, the questions in the second column are considered of equal importance. Figure 16.1 below illustrates a methodology with four additional phases that would take into account these and other requirements of educational software.

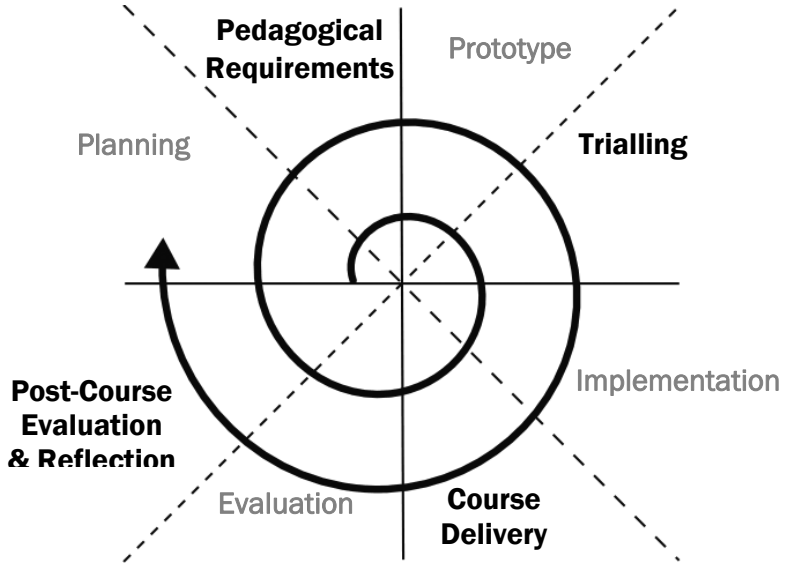

Figure 16.1: The spiral-Ed eLearning software development methodology

Figure 16.1 visually represents the proposed spiral-Ed software development methodology. It can be read as a spiral of activity traversing four stages, each of which has two parts. Each iteration of the spiral may take a year or more since some of its milestones have long user cycles, particularly in formal education environments where courses are only taught once a year. Our methodology provides guidelines for the project team to follow throughout the software development life cycle of the eLearning application. It can be integrated into software development methodologies for large projects, especially where there are greater resources at hand, or it can be used on its own for smaller projects. The fact that it is based on extensions to the industry-standard spiral methodologies, means it can be incorporated more smoothly and easily by engineers into the development process.

Figure 16.1 presents an overview of the methodology. The spiral conveys that it is an iterative process, made up of engineering and educationally driven stages. The engineering stages are: planning, developing a prototype, implementation and evaluation. Mirroring each of these stages, are the proposed educational stages: assessing the pedagogical requirements, trialling, course delivery and post-course evaluation and reflection.

The four stages in the methodology each have two parts, the first of which is standard to traditional spiral models and the second of which is specific to spiral-Ed:

Planning. During this phase, planning for the project commences, software requirements are gathered, and a design for the system is produced.

> *Pedagogical requirements:* Pedagogical requirements must also be gathered. These are obtained from the teacher and include the nature of the learning activity, the underlying pedagogy, how the activity will be assessed and any previous feedback from students about the learning activity which might help inform the software developers. The project team must clarify with the teacher what the students are expected to learn, and how. At this stage, the post-course evaluation of the student experience must also be planned.

Developing a prototype. This is a mandatory step in any engineering project. A prototype is normally built and used for testing and is accompanied by a risk analysis of the project meeting its outcomes.

> *Trialling:* The outcome of this stage is to test the prototype with a focus group of students. Trialling can include both teaching staff and students. Students' perceptions of the software and their approaches to the learning activity should be evaluated. The prototype must provide the core functionality that is required to facilitate a learning activity.

Implementation. The system design is fully implemented to produce a functional system. The software is also tested for quality assurance.

> *Course delivery:* At the end of the implementation and testing phase, the software is used within a course, where students are expected to use the software to achieve specific learning outcomes on which they will be assessed. During this phase, information about how students approach their use of the software should be obtained, where possible.

Evaluation. This phase evaluates the software application outcomes of the project to date before the project continues to the next iteration.

> *Post-course evaluation and reflection:* Our methodology requires a post-cycle evaluation and reflection to be included. During this phase the data collected for the student's experience is analysed, and recommendations from the results are reported.

The inclusion of a course delivery testing phase in the model has proven particularly useful for the projects in which it was included. Evaluating the software within a real course allows for the project team to understand the students' perceptions and experience of the software, and how it is actually being used to shape their attitudes towards learning. The main advantage for testing the software within a course is that learning generally takes time, and several weeks are required to test whether or not the software has had a positive impact on student learning. For example, pedagogy such as reflective learning strategies, requires the learner to first understand the knowledge content, and then spend time reflecting on what they have learned. It is unrealistic to believe that this can be achieved in a short time frame, or in an inauthentic lab testing environment.

Furthermore, testing the software within a course for the duration of a semester will uncover social phenomena over time. For example, in Project 2 discussed below, it was discovered that a few isolated students used a collaborative feature of an eLearning system to submit plagiarised work. The act of plagiarism would rarely occur in a focus group environment, as test subjects would be aware that their responses and actions would be closely monitored. Therefore, our methodology includes a trial period of the eLearning software within a real-time course environment to ensure the project team can observe how students use the software, (and sometimes may abuse the software) in ways that can only unravel within an authentic space.

Application and evaluation

The methodology described here has been used for two educational software development projects: *Beehive*, a system which allows teachers to use educational design patterns to build on-line activities for groups of students; and *dotFolio*, an

electronic portfolio used to develop students' reflective and writing skills. The emphasis in this section is to demonstrate the spiral-Ed methodology as it might be used by describing how each phase was carried out and how resulting information about the student experience influenced the development process. The software applications are currently in different stages of development.

Project 1 – Beehive: Enabling teachers to design on-line group work activities
Beehive is an educational software application that enables teachers to design and build collaborative learning activities on-line (Turani & Calvo, 2006; Turani et al., 2005). Teachers are able to use the eLearning tools provided by Beehive to design synchronous collaborative learning activities. For example, if we take the topic 'Making your backyard pool more sustainable', a teacher could:

- identify the purpose of the task and its desired outcome;
- identify a sequence of learning activities leading to the outcome;
- choose eLearning tools in Beehive to support these activities;
- develop the sequence of activities using the tools and help students understand the topic through them.

In the example of 'Sustainability', the eLearning tools chosen could include an instruction tool to explain the purpose of the task, a brainstorming tool which requires groups of 4-5 students to share their ideas about sustainability on-line, a voting tool, which requires the students to rank their shared ideas in terms of their effectiveness for the topic of sustainability, a discursive tool which requires the best two or three ideas to be more fully described by the students and a plenary tool which enables students and the teacher to provide conclusions and closure to the activity. Each of these tools has the potential to include audio and video resources to increase the richness of the experience.

During the development of Beehive, the following four educationally-focused phases were used to inform the software development process, as described below.

Pedagogical requirements. The goal of Beehive is to provide teachers with a tool that helps them to design collaborative learning experiences, and support them in the design task with research-based scaffolding in the form of educational 'patterns'. The functional requirements were based on other pattern-based systems (McAndrew, Goodyear & Dalziel, in press).

Trialling. A prototype of the application was developed using a combination of technologies, including the dotLRN Learning Management System and Flash Communication Server. The prototype was trialled with a group of seven users (tutors and postgraduate students). The participants were explained the purpose of Beehive, given a mock activity, organised into three groups, and allocated to different computers in separate rooms. After the mock activity, all participants were asked about how they had used the tool, what advantages and difficulties they found, and how they would use it in tutoring situations. The results revealed certain usability problems that were addressed at this stage.
Course delivery. The Beehive software program was used by a group of postgraduate students enrolled in the teacher training course 'EDPC5021, Introduction to the Learning Sciences', part of an Education Masters program.

Post-course evaluation. To evaluate their experience of using the software, the trainee teachers were asked what they thought the purpose of Beehive was, how they went

about using it to design student learning activities and why they approached using Beehive in the way they did. This phase in the development process proved essential as it revealed that the software was falling short of meeting some of its educational goals. For example, when asked what they thought the purpose of Beehive was, some trainee teachers responded like this:

> Beehive is for developing the skills of students in the group, and allowing teachers to capture reusable learning objects.
>
> To provide patterns for on-line teaching in an accessible way and to automate processes of student group work.
>
> Model and structure face to face learning activities in an on-line way and provide a tool to manage on-line classes.

When asked the same question, other trainee teachers in the same class said:

> To enable students to work collaboratively over the internet. To share ideas and come up with group decisions.
>
> To facilitate on-line collaborative learning sessions to promote student understanding.

While the first group of quotations are not necessarily inaccurate (although the concept of trying to transfer a face-to-face activity to the on-line context suggests an undeveloped understanding about the way on-line learning can help student learning), they all share a focus that is on something other than students. The designers of Beehive first and foremost wanted a tool to improve the effectiveness of student learning. Additional benefits such as reusable objects, ease of design and managing the on-line environment are secondary. In contrast, the second group of quotations showed an awareness that the main purpose of Beehive was to enable students to share ideas, and to 'promote student understanding'.

When asked how they approached using Beehive, the same first group of trainee teachers responded:

> As a trainee teacher, I sit, watch, and do what I am told with the tools in Beehive.
>
> I just pick the tasks I'll need, set the scenario and information and slides and sequence the rest of the tasks.

The second group of trainee teachers said things like:

> I prefer to do some preparation about student needs before I use the program. I like to have the script written before I go to Beehive.
>
> I like to design learning activities in relation to student learning outcomes, especially when selecting the pedagogical techniques and selecting the tasks.

The software developers noticed a consistent theme in the comments made by the trainee teachers in relation to their approach to using Beehive. Comments such as those in the first group tended to focus more on the software itself, 'the tools in Beehive' or 'set the scenario and information and slides'. They did not tend to display any

awareness of the learning context in which the on-line tools would be used. In contrast, the second group of comments about teacher approaches to using Beehive foregrounded the educational context. In the first comment, emphasising 'student needs' and in the second quotation, 'student learning outcomes'.

We were then able to take the trainee teachers experience into account for the next development cycle. New ways to engage them in the development process as a flexible, student-centred activity were added. The changes included adding the possibility of students and teachers being able to co-design the activities, and a multimedia animation that demonstrates how Beehive affects students' learning, to help the users better realise potential student-centred benefits of the tool. The co-design feature allows students to engage better in the activity, and increase awareness of their own learning process as they understand better how the activity structure affects them. Including the educationally driven phases into our development process, was essential in truly understanding whether the software was not just usable or technically stable, but whether it was meeting its learning goals.

Project 2 – DotFolio: Enabling students to reflect on their learning

Portfolios of student work have often been used to indicate the quality of what students have learnt to interested parties and to facilitate student learning through reflection. When portfolios are enabled electronically, the effectiveness of retrieval and presentation of student work encourages a broader usage of e-portfolios as a part of students' everyday learning experience (Roberts, Newble & O'Rourke, 2002). Within higher education a growing body of research is identifying the value of e-portfolios for developing students' reflective learning practice (Roberts et al., 2002).

The goal of the dotFolio project is to investigate how students can use electronic portfolios for the purpose of learning, particularly through reflection (Carroll & Markauskaite, 2006). To facilitate this research goal, an open-source e-portfolio application - called 'dotFolio'- has been developed, based in part on investigation into the student experience of the software.

Undergraduate engineering students have been using dotFolio to reflect on current issues of engineering interest. For example, on the topic of Occupational Health and Safety, students would:

- select a news story from a source such as the *Sydney Morning Herald*;
- link it or upload it in dotFolio;
- summarise their understanding of the significance of the issue and its importance to the engineering profession and write their reflections in the online log ('blog').

The four education-centred phases used to inform the development process included:

*Planning***.** The system was first used by engineering students enrolled in the first-year unit of study Professional Engineering. The first development iteration of the dotFolio project was based on the basic requirement of replacing a paper-based report writing activity with an online version, with the intention that this would improve the benefits of the activity for students. The students were to develop their reflective practice skills (as in the similar paper-based activity) as well as other generic graduate attributes.

Trialling. A prototype was built and then shown to a group of tutors and students. Based on the test results of the first prototype, the team decided to limit the functionalities available, so they would be more clear to users. In this way, the tool

became more transparent and simple to both academics and students. Other usability issues became evident in this phase, as did issues around how to reinforce the idea to students that no copyrighted materials should be uploaded. Tutors who were going to grade assignments submitted through the system required a grouping functionality that would allow them to mark more efficiently.

Course delivery. About 260 first-year engineering students used dotFolio during one semester (16% female, 84% male).

Post-course evaluation. To investigate the students' experience of dotFolio and of reflective learning, an evaluation questionnaire was developed and a two-phase survey was conducted (Carroll & Markauskaite, 2006). The questionnaire was administered at the beginning and end of the semester. The first group of items interrogates student perceptions of the technology in their learning experience. The second group of items interrogates their approaches to using the technology. The last item investigates student conceptions of reflection. Data was collected twice in order to see how attitudes changed. Then, the significance of changes in students' attitudes were analysed using the paired-samples t-test (and marginal homogeneity tests).

The study showed how, by the end of the semester, students had a more positive view of the technology, but more negative view of collaboration with peers. No significant changes were observed in students' attitudes towards reflective learning in engineering.

Students strongly disagreed with some closed-ended questionnaire items they were asked to complete about the value of collaboration, 'My classmates feedback on my entries in the logbook will help/helped me to achieve the learning outcomes ' and 'Reading and commenting on my classmates' entries in their logbooks will improve/improved my understanding of current professional engineering issues'. In addition, the open-ended responses of some students indicated that they felt inherent conflict between collaboration with peers and the privacy of their reflection. This aspect of the students' experience revealed in this stage of the development process, prompted changes in the software, such as the addition of a functionality allowing students to make their postings private (not visible to anyone else except the tutors).

In addition, when students were asked in an open-ended question to write their concerns about the dotFolio tool, 12.7% at the beginning of the semester and 25.3% at the end of the semester indicated that plagiarism was an important one. This result provided insight into reasons for the students' dissatisfaction with collaboration. The plagiarism issue was also addressed by changes in the software. A plagiarism detection system that allows teachers to do a 'collusion' analysis and checks for submission similarities was implemented in dotFolio (Garcia Adeva, Carroll & Calvo, 2006).

By the end of the course, students were the most positive about the functional features of dotFolio. Some student comments indicated that they appreciated practical benefits: 'the ability to store files in a central site. This also helps with the transfer of data from home to university, as well as sharing of files between group members in group work'; 'much potential, system that I believe would reduce workload and streamline some assignments.' Other students described the benefits as being closer to their learning experiences: 'get to be exposed to real engineering projects'; 'it's different. I'll learn new aspects of computing…'

However, the students' answers to the items about reflective learning in engineering, and the lack of significant changes in their attitudes during the semester indicated that students did not relate the work they had done in dotFolio to their engineering practice.

As a way of addressing this issue with the software, a new feature integrating the e-portfolio system with the faculty unit of study database, was added to dotFolio (Calvo, Ellis & Carroll, 2006). This function now allows students to link the description of the graduate attributes being developed, with their entries in dotFolio. The results of these changes may be tested during another iteration of the spiral.

Conclusions

In this chapter, we have sought to present a software engineering methodology that is informed by the learning experiences of students in higher education. The methodology is an adaptation of recognisable engineering methodologies referred to as spiral approaches to development, but extended for educational software to include stages for obtaining an understanding of how students think about, perceive and approach using the software for learning.

The spiral-Ed methodology proposed has been discussed in the context of two learning software development projects: Beehive, a synchronous learning design tool and dotFolio, an electronic portfolio tool. These projects have provided a way of discussing how the students' experience of learning using these technologies can be fed back into the software development process to improve the quality of knowledge upon which the software systems are developed.

It is clear from the experience of the authors, that supporting students' learning experiences effectively with learning technologies is a complex goal. To do so in an informed and principled way, specialist knowledge from different disciplines is required and significant effort is necessary if the knowledge is to be synthesised and applied in a useful manner. Although the methodology described here is in its early days of development and application, our experience so far has been very positive, and has been valuable from the perspective of both the software engineers and the educational researchers. We argue that truly effective and supportive learning software can only be developed if some understanding of the experience of learning supported by the software itself is included in the development process. We anticipate that if software development teams increasingly employ a development methodology (such as the one proposed here) that is informed by the essential educational aspects of the software being produced, that the effectiveness of eLearning technologies overall will be significantly improved.