An Integrated Tool Set to Support Software Engineering Learning

Anne Philpott, Jim Buchan and Andy Connor
Software Engineering Research Laboratory, Auckland University of Technology
{aphilpot, jbuchan, aconnor} @aut.ac.nz

Abstract

This paper considers the possible benefits of an integrated Software Engineering tool set specifically tailored for novice developers, and reflects on the experience of having software engineering students produce various components of this tool set. Experiences with a single semester pilot are discussed and future directions for refining the model are presented.

1. Introduction

A software curriculum should involve five complementary elements, namely principles, practices, application, tools and mathematics [1]. The Software Development major at AUT addresses these elements throughout a number of interlinked papers, each of which has a different focus in preparing undergraduate students to be software engineering practitioners, as illustrated in Figure 1.

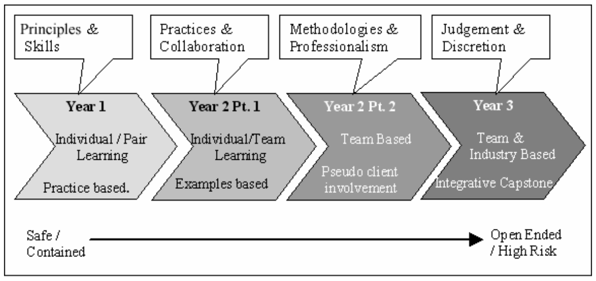

Figure 1. Software Development Skill Acquisition

The Software Engineering paper is taught in the fourth semester of an undergraduate degree in computer and information sciences. Students entering the course have previously completed courses in programming, software design and implementation, data and process modelling, and database design. They therefore have experience with java programming, the use of UML in modelling, SQL and the design and construction of databases. The Software Engineering paper has a focus of reinforcing and applying the principles and practices introduced in preceding papers in a simulated project environment using a constructivist learning approach.

The intention is to draw in the tools element identified by Meyer [1] by investigating both the role a purpose designed tool could play in this approach and the implications of using the tool development as a learning activity. The aim is to promote learning through this activity, without the students being explicitly informed that they should be learning certain principles and best practice. This approach has been applied in introducing concepts of requirements engineering into software engineering curricula [2].

2. Integrating Tools and Learning

In the constructivist view, new learning is actively constructed by the learner through interaction with the environment. It incrementally develops from previous experience and knowledge. As their current concepts and knowledge are challenged by new experiences, new cognitive structures emerge. Knowledge creation occurs as these new mental models, attitudes and schema are internalised by the learner when they make and evaluate decisions, make and test hypotheses, and take and reflect on actions. [3, 4, 5]

Such a view of learning raises the question of how to best foster a learning environment that supports such an approach to constructing software engineering learning. There are a number of pedagogical principles, strategies and practices suggested in literature which have guided our design of the learning experiences in this programme [3, 4, 5].

Table 1 illustrates how these principles have been specifically implemented in the Software Engineering paper and identifies the implications for a tool set that would support these principles.

Constructivist Principle	**Software Engineering Paper Implementation**	**Implication for Tool Set**
Authentic tasks. The process of constructing strongly-linked knowledge requires a certain type of problem-solving. The problem should be interesting, and realistic with elements of uncertainty, ambiguity and complexity.	A problem-based learning approach is adopted [6]. Understanding of the problem evolves through learner interaction with a client with a software development need. Solving the problem (developing the software and managing the process and quality) is the motivation for new learning. Students undertake activities rather than listen to lectures. Lecturers are referred to as “mentors” and are collaborators and facilitators of learning. Resources (by way of reference material and access to experts) are provided as activities demand.	Tools should be useful in supporting/guiding students in the diversity of activities related to solving realistic problems in software engineering, but should not overwhelm the activities. For example, easy to update working artefacts, provides readily accessible information that informs real decisions, allows monitoring of progress and quality.
Build on learners’ prior knowledge. Students’ current knowledge of developing a software solution should be evaluated and the learning activities should make use of skills, concepts and language that have some familiarity to the students.	Based on previous knowledge of development activities and tools and extends activities to include a more extensive part of the development life cycle, for example: more complexity in domain understanding, client management, and planning and project management.	The tool should be flexible in the language used and adaptable to a number of practices from different development methodologies. For example, previous knowledge may include multiple practices for the same activity (eg. User stories / use cases)
Problem solving activities should reinforce useful knowledge such as best practice and challenge misconceptions and poor practices.	Examples include such things as having the client change requirements to reinforce designing for change.	The tools could guide and/or restrict users to good practice and trigger questioning of poor practice. For example the requirements management tool could assume an acceptance test is associated with every user requirement.
Construction of learning should include social interactions and collaboration	Team-based development, though this is continually a challenge as students have conflicting commitments. Team-building exercise. Multi-team sharing of learning.	Tool should support and promote collaboration both when team members are co-located and when separated or non-synchronous.
Support higher order thinking such as analysis, critical thinking, reflection, self-evaluation, and conceptualisation.	Mentor role to ask probing questions. Peer reviews. Opportunities for reflection provided by a reflective team presentation, and individual reflective report.	The tool should be simple to learn and use and not interfere with thinking about the software development activities. The tool should support analysis, reflection and abstraction.

Table 1. Application of Constructivist Principles to Software Engineering Course and Tools

Some studies have shown that students have to learn and manage new tools when learning Software Engineering. Whilst not perceived as a challenge [7], this has in reality been observed to detract from learning principles. Students are often confounded by the complexity of commercially available tools which offer considerably more functionality than is required. There is clearly a need for a suite of lightweight tools that are fit for purpose, with a common interface that could reduce the burden of learning complex tools.

Much of the focus to date on developing appropriate tools to facilitate learning in Software Engineering teaching has been on integrated development environments [8]. In addition, some tools have been introduced to Software Engineering course that are intended to reinforce good practice in group work [9] though these have been found to be treated as an isolated section of the course and have not been adopted by the students in their formal group activities. Other tool use includes the use of standard groupware products to support distributed projects [10] but there has been little work done in the area of tools focused on the Software Engineering Process, particularly those aspects which are known to be hard.

3. Scenario Description

A key element of the Software Engineering paper is a simulated group exercise, where a member of staff presents a need for a software solution and plays the role of the client with whom the students must interact. The purpose of this exercise is to assist development of an understanding of the spectrum of software development methodologies. There is a particular emphasis on communication and client interaction approaches, requirements elicitation practices, project management and risk management activities. The outcome of the exercise is the delivery of an agreed software product to an agreed deadline. Students work in teams of 3-5 members and have face-to-face interaction with the client for their project.

The aim is to make the scenario realistic, fun, critical and accessible. These characteristics are amongst those identified by Stiller and LeBlanc [11] as being necessary to ensure that Software Engineering is taught effectively. Previous semesters have utilised a wide range of application domains.

In the current semester, a member of staff presented a need for lightweight tools to support learning of software engineering principles in an academic programme. Rather than playing a role, the staff member was providing a realistic situation. The intention was to utilise students enrolled in the paper to develop an initial set of tools for project and requirements management that would be appropriate for use in the teaching of principles in a software development programme.

4. Discussion

The realistic scenario has provided a useful experience from which to further enhance the delivery of the paper. In the first instance, despite having briefed the student teams that the need for developing the tools was to promote learning on Software Engineering papers, none of the student teams actually adopted process tools that supported their development activities. This supports the belief that commercial tools are overly complex for use in the exercise. It was also observed that students' analysis of the problem domain tended to be superficial, with little evidence of critical thinking and abstraction. In particular, none of the student teams really delved into, for example, trying to understand what software requirements are and how they need to be managed. Students tended to adopt a requirements representation that had been presented in preceding papers without questioning whether an alternative representation was appropriate.

In general, the temptation for students was to jump to familiar approaches no attempt to identify and evaluate alternative solutions. As a result, students found it difficult to manage changing and refinement of user requirements despite accepting this was inevitable. Similarly, managing the client and dealing with ambiguity and uncertainty was a big challenge for the students. There is a possibility that the demands for developing complete tools was too high, forcing the students to cut back on activities that distinguish Software Engineering from writing code.

In the future, a tool set with rudimentary functionality will be introduced at the beginning of the paper. Each successive group of students will refine and extend the tool set. This approach has been used in other Software Engineering courses with some degree of success [12]. The approach introduces the ability to emphasise the principles of reuse and refactoring. It will also expose students to having to work with code developed outside of their peer group. The challenge and benefits of running consecutive projects on the same topic and as a result working with other people's code has been identified as having distinct benefits [1].

5. Conclusions

Whilst Meyer's [1] contention that software engineering students should be exposed to tools that are currently used in industry is not disputed, it is important to ensure that such exposure is not to the detriment of a focus on developing understanding and awareness of principles and practices. An initial novice tool set that enhanced this focus might assist in developing enough understanding that the complexity of any selected current industry tools later introduced would not overwhelm their purpose.

In addition such a tool set could also support the 'inverted curriculum' approach suggested by Meyer. Use of relevant parts of the tool could be introduced to support earlier papers with 'progressive opening of the black boxes' resulting in the software engineering students enhancing the tool suite as their project.

6. References

[1] B. Meyer “Software engineering in the academy”, Computer, 34(5), IEEE, 2001, 28-35.

[2] D. Callele. and D. Makaroff 2006. “Teaching requirements engineering to an unsuspecting audience”, Proceedings of the 37th SIGCSE Technical Symposium on Computer Science Education, March 3-5, 2006, (Houston, Texas, USA), 433-437

[3] S. Hadjerrouit “A constructivist approach to object-oriented design and programming”, Proceedings of the 4th Annual SIGCSE/SIGCUE ITiCSE Conference on innovation and Technology in Computer Science Education, June 27- 30, 1999, (Cracow, Poland), 1999 171-174

[4] M. Ben-Ari “Constructivism in computer science education”, SIGCSE Bulletin, 30(1), 1998, 257-261

[5] S. Hadjerrouit “Toward a Constructivist Approach to E-Learning in Software Engineering”, Proceedings of World Conference on E-Learning in Corporate, Government, Healthcare, and Higher Education, November 7-11 2003, (Pheonix, Arizona, USA), 507-514

[6] J. Armarego “Advanced software design: a case in problem-based learning”, Proceedings of the 15th Conference on Software Engineering Education and Training, February 25-27 2002 (Ottawa, Canada), 44-54

[7] M. Gnatz, L. Kof, F. Prilmeier and T. Seifert “A practical approach of teaching Software Engineering”, Proceedings of the 16th Conference on Software Engineering Education and Training, March 20-22 2003, (Madrid, Spain) 120-128

[8] C. Kelleher and R. Pausch “Lowering the Barriers to Programming: A Taxonomy of Programming Environments and Languages for Novice Programmers”. ACM Computing Surveys, 37(2), 2005, 83-137.

[9] C. Liu, “Using Issue Tracking Tools to Facilitate Student Learning of Communication Skills in Software Engineering Courses”, Proceedings of the 18th Conference on Software Engineering Education & Training, April 18-20 2005, (Ottawa, Canada), 61-68

[10] O.P. Brereton, S. Lees, R. Bedson, C. Boldyreff, S. Drummond, P. Layzell, L. Macaulay, and R. Young “Student collaboration across universities: a case study in software engineering”, Proceedings of the 13th Conference on Software Engineering Education and Training, March 6-8 2000, (Austin, Texas, USA), 76-86

[11] E. Stiller and C. LeBlanc “Effective software engineering pedagogy”, Journal of Computing in Small Colleges, 17(6), 2002, 124-134

[12] C. Mingins, J. Miller, M. Dick and M. Postema. “How We Teach Software Engineering”, Journal of Object Orientated Programming, 11(9), 1999, 64-69