Increased practical orientation in a Software Engineering course

Andreas Rösel
ABB Corporate Research
Postfach 101332
D-69003 Heidelberg
roesel@decrc.abb.com

Abstract

Software Engineering is viewed as an engineering discipline concerned with bridging the gap between appropriate science knowledge and software applications. Curricula for traditional engineering disciplines include a substantial amount of practical and project work. We report experiences with increasing the laboratory and project component for Software Engineering Education at the Queensland University of Technology. Particular emphasis is given to the project-based Software Engineering subject. In the teaching of this subject some research findings on lasting knowledge for Software Engineering were applied. The increased practical orientation is seen as a success, but several important lessons about „how not to do it“ had to be learned.

1. Introduction

Software Engineering bridges the gap between computer science and application know-how. This science / application bridge is a typical characteristic of engineering disciplines. The education of engineers in more traditional branches of engineering include substantial amounts of practical and project work [1]. Educators in software engineering are considering to which extent the practical components are helpful, maybe even essential, for successfully conveying a sound foundation for a career as a professional software engineer [2, 3].

The knowledge conveyed in a University education should provide a sound foundation for a continuous learning process during a professional life spanning some decades. Tough choices have to be made about the topics and the depth of coverage. In particular in the still -evolving discipline of software engineering, principle foundation knowledge may quickly get lost in the vast amount of (always changing) operating systems, programming languages, methods, etc. In [4] a systematic approach has been taken to distinguish between lasting know-how and the many variants of changing know-how. The more abstract activities underlying software engineering, namely engineering, design and problem solving, contain principles which are of lasting value and are found to be applicable to software engineering. They also help to identify (more specific) principles in the domain of software engineering - which are likely candidates to be considered in the design of software engineering curricula.

2. Typical characteristics of SE problems

In the effort to most efficiently convey a lasting foundation for software engineering expertise the question arises as to what and how much practical work is to be included in the curriculum. Practical work at the level of isolated small problems can not be sufficient. On the other hand, engineers, in contrast to apprentices, will not be trained over a long period through learning by doing on one project. Rather, the lasting theory know-how must be complemented by appropriate project experiences.

Table 1 illustrates the limitations of ‘toy problems’ in the education of software engineers. For educational purposes, a task may be a real world task or an arificial one. However, it must exhibit the characteristics appropriate for illustrating the principles to be conveyed.

3 Practical work in the QUT course

In 1991 a new course for the education of software engineers was developed at the Queensland University of Technology (QUT) Brisbane, Australia [5]. Here we are interested in the amount and organization of practical work in the form of a continuous strand of laboratory and project subjects.

Team	individual effort	group effort
Requirements	fixed requirements	negotiated requirements
Documentation	documented after completion	working documents
Duration	short spurt of effort	ongoing effort
Responsibilities	follow solution steps	make vital decisions
Customer	no customer	customer waiting
Result	problem solution	complete package

Table 1: Typical characteristics of toy problems versus real world SE problems

The course is located in the Faculty of Information Technology. For historic reasons the course was limited to a three year duration. The course designers had experience with predecessor courses at the same university and similar courses at other universities. The course has a foundation component and allows for specialization in more business- or technical-oriented software engineering. Each semester has a practical component taking up one fifth to a quarter of the total semester effort. In the first half of the course the practical component is in the form of Laboratory work.

Characteristics of Lab - work

Team	some group, some individual effort
Requirements	fixed requirements
Documentation	documented after completion
Duration	spurts of effort, some follow-on tasks
Responsibilities	follow solution steps
Customer	no customer
Result	problem solution

Advantages

- Practical component made explicit.
- Coordination of prac workload.
- Efficient booking of laboratories.
- Tutors available in laboratories.

Disadvantages

- Practical work and theory not always synchronized.
- Temptation to off-load practical part to assistants.
- Big laboratories encourage plagiarism.

In the third year the practical components take on the form of projects. The strand of practical work initially provides a lot of guidance and has more the characteristics of 'toy problems', while in later semesters more and more responsibility is shifted to the student to be educated as a software engineer, and the characteristics of the practical work resemble more those of application problems.

Characteristics of SE Project

Team	group effort (3-4 people),
Requirements	negotiated requirements
Documentation	working documents
Duration	ongoing effort
Responsibilities	make vital decisions
Customer	customer waiting
Result	complete package

Advantages

- Real world characteristics provide means for conveying lasting principles.
- High learning value.
- Many non-technical aspects are included.

Disadvantages

- High workload for students and teaching staff.

An overview of our experience with the strand of practical work is given in the form of advantages and disadvantages (see above). The next focus will be on experiences with the project-based subject Software Engineering.

3. Software Engineering - Project Subject

This project subject provides a guided experience through one full development cycle of a software project during one semester. Parallel to the practical work an integrated lecturing part provides essential theory (which is not covered by prerequisite subjects) and ensures the learning experiences intended are prepared and wrapped up. In the practical part, students work through a streamlined software engineering project. The structure of this subject was based on experiences from various sources [e.g. 6, 7]. It was monitored closely through two reviews per semester (one informal, one formal) and improved accordingly. Some key features are:

Group work

Students work in teams of three to four people. They are presented with organizational models and are free to choose their team structure.

Contributions to deliverables are made explicit. Problem resolution is to happen at a team level. For crisis management teaching staff is available to play the role of a next level manager.

Weekly project meetings during tutorial time

For part of this time a lecturer or an assistant is available to play the role of the customer (e.g. for clarification of requirements), consultant (for checking if this design is feasible), or reviewer. During project meetings teams are encouraged to provide drafts of deliverables (e.g. a certain document) for an informal review. Deliverables are also submitted and reviewed during these meetings. The review has one of the results: accepted, accepted with minor modifications or rework. The project teams get immediate feedback (!), get a live demo of how a review is conducted and (maybe most importantly) absorb the value of technical reviews.

Staged development

Each project team has to define the scope of a minimal as well as a standard system. The team members are confronted early with the task to distinguish between essential features, consider risks and the need for a fall-back position. The teams know that the minimal system is expected to be delivered even if unforeseen setbacks occur (e.g. a team member gets sick). It is our experience that teams set ambitious goals, even after being told that previous teams were too ambitious. Less than half the teams usually complete their planned standard system. But most teams are able to deliver at least the minimal system. (Note that functionality makes up only a minor part of the final grade.)

Selected documentation

Deliverables are parts of documents for each development phase which are of essential value for the project progress. A typical table of contents for relevant documents is provided, and certain sections are to be completed as a deliverable. For the requirements document this included the prioritized feature list for a minimal and for a standard system. In the test documentation this included from each team member the test plan for one module or feature (limited to a few pages) and later the test report documenting the running of tests according to that plan.

Availability of templates and examples

For each deliverable, templates are available, as they should be in any professional software development organization. Also, examples of deliverables from other projects are made available for comparison including the remarks made during reviews. Key approaches for each phase and typical difficulties encountered are presented.

Implementation and acceptance test

It has been found valuable that the projects have an implementation phase. Many of the learning experiences occur at a deeper level because of the experiences in this phase. However, some may not occure at all, should this phase be omitted. Note, that similar to the provision of templates at the code level at least one component is provided. This leads to a reuse experience and reduces the development effort. The delivered system (for example a simple mail system or version control system) is tested by the staff member acting as the 'customer'. Part of the acceptance test is to follow the instructions from the partial users manual (another deliverable).

Reflective project execution

Projects can take on a special dynamic of their own. The continuous interjection of preparation, meetings, feedback, wrap-up is vital for keeping a reflective attitude awake for all involved. This is also signified by the last deliverable of the project, a project evaluation covering people aspects, technical aspects etc. Here each project team member noted their individual perception of major project influences, for example significant decisions or mistakes and personality issues.

Feedback / Criticism

Critically we have to note that the software engineering subject was somewhat overloaded. (In a three year curriculum a large number of what one would consider essential software engineering topics are unlikely to be covered elsewhere). Despite conscious efforts to leave out certain topics and focus on parts of deliverables the workload was significantly above average. It may be necessary to identify and then concentrate more rigorously on those key topics of software engineering which are best presented in a guided project approach. Other vital areas are to be covered to a sufficient degree elsewhere in the course.

As positive we found that it was possible to provide in a third year project-based subject an efficient and most valuable learning experience in Software Engineering. A higher ratio of tutorial staff was needed to provide the required guidance for the project teams. Feedback from student surveys showed they rated this as a hard but important subject which they recommended to take even if it wasn't compulsory. Employers were reported to be most interested in the contents and execution of the team project during job interviews. Lecturers offering graduate and post graduate projects provided another (unexpected) source of feedback. Several reported that for certain projects the quality increased noticeably and that documentation was handed in which had not requested. It was found these were projects done by students who had completed the guided software engineering project.

4. Outlook

If software engineering is recognized as an engineering activity then the course design should also reflect this, particularly in terms of the systematic use of practical and project-based learning. Course designers will continue to face the challenge of how the various types of engineering know-how are best acquired in such settings and how such experiences can build on each other most effectively.

A significant part of the learning experiences in a project-based software engineering course are fundamental engineering, design and problem solving experiences. It may be worthwhile to consider how these can explicitly be identified as such. As fundamental know-how it can be expected to be long term applicable, more transferable and valuable. An interesting example aiming in this direction is project work as part of a cross-faculty introductory engineering subject [8].

The decisions about the contents of practical and project-based software engineering subjects are also influenced by the length and type of the course, for example: a duration of 3 or more years, a sandwich type course with an industrial component. We noted a higher level of learning when teaching the same project-based software engineering subject to graduate diploma students who had previous industrial experience. This can be seen as an indicator for the value of need-based University courses for people practising in software engineering. As the need and value for continuous education is increasingly being recognized [9] this option might be particularly noteworthy in combination with shorter first degree courses - concepts of interest to students, employers as well as educational decision makers.

Summary

The experience at the faculty of Information Technology at QUT showed that an increased and explicit practical component can help to better convey the foundation of software engineering expertise. Valuable project subjects must exhibit the characteristics of typical software engineering application problems, but must not necessarily be real applications. In particular it was found valuable to have a guided experience with selected aspects from a complete development life-cycle before students embark on more loosely supervised projects.

References

[1] IEEE (1983): The 1983 IEEE Computer Society Model Program in Computer Science and Engineering, IEEE Computer Society, L.A. California.

[2] Bailes P. A., Roesel A. and Salzman E. J. (1989): *A Proposal for a Bachelor's Degree Program in Software Engineering*, In Gibbs N.E. (ed.): Software Engineering Education, Lecture Notes in Computer Science, Vol. 376, Springer Verlag, Berlin.

[3] Tucker A.. B. (ed.) (1991): Computing Curricula 1991 - A Summary of the ACM/IEEE Joint Curriculum Task Force Report, Communications of the ACM, Vol. 34, No. 6.

[4] Roesel A. (1991): Perpetual Truths of Software Engineering, Masters Thesis (Computer Science), The University of Queensland, Brisbane.

[5] QUT (1991): Queensland University of Technology Handbook, QUT, Brisbane.

[6] Lamb, D. A. (1988): Software Engineering - Planning for Change, Prentice Hall.

[7] Sommerville, I. (1989): Software Engineering, 3rd ed., Addison-Wesley.

[8] Rubinstein N. F. (1980): *A Decade of Experience in Teaching an Inter-disciplinary Problem Solving Course,* In: Tuna D. T. et al. (eds.), Problem Solving and Education: Issues in Teaching and Research, Wiley and Sons, N. J.

[9] Humphrey, W. S. (1995): A Discipline for Software Engineering, Addison-Wesley.

About the author

Andreas Roesel is a Software Engineer with 14 years of industrial and lecturing experience. He has worked as Software Engineer for Westinghouse, as consultant and lectured in Object oriented Software Development and Software Engineering at the Queensland University of Technology (Brisbane, Australia). He has an engineering degree and a Master of Science in Software Engineering. Since 1993 he is working as a consulting engineer for ABB companies in the field of Software Engineering.