Implementation of a Software Engineering Course for Computer Science Students

Ivica Crnkovic
Department of Computer Engineering
Mälardalen University
Box 883, 721 23 Västerås, Sweden
+46 21 103183
Ivica.Crnkovic@mdh.se
http://www.idt.mdh.se/personal/icc

Magnus Larsson, Frank Lüders
Research and Development
ABB Automation Products AB
721 59 Västerås, Sweden
+46 21 342666
{Magnus.Larsson | Frank.Luders}@mdh.se
http://www.idt.mdh.se/personal/{mlo | fls}

ABSTRACT

Experience from industry shows that graduates in computer science generally lack many of the skills required in software development projects. This presents a challenge to academic institutions. This paper describes our experiences in implementing a course in software engineering at a Swedish university. A set of challenges is presented and it is described how these were met using a combination of lectures and project work. The results of the projects, the lessons we have learned, and the feedback from the students are discussed.

Keywords
Software engineering, Education.

1 INTRODUCTION

The need for software engineers is growing and it is more important than ever to educate qualified personnel for the software industry. Reports from industry show that, although graduates in computer science are generally very knowledgeable, they often lack many of the skills needed in software development projects. This presents a challenge to academic institutions to incorporate training in such skills in their curricula.

This paper describes our experiences in implementing a one-semester course in software engineering for computer science students at Mälardalen University in Sweden.

2 EXPERIENCE FROM INDUSTRY

During many years of working with Swedish industry, we have observed on almost regular pattern of capability exhibited by graduates in computer science or software engineering, and we have noticed the same in students working in industry as part of their thesis projects or in summer employment. Newly graduated engineers and students usually have knowledge of different programming techniques. They learn new languages quickly, and are soon familiar with new technologies. They solve problems quickly, they can take initiative and soon actively participate in working teams. However, they show significantly less of the understanding, motivation and ability called for by the software engineering disciplines. The students generally have:

- difficulty in with writing good technical documents;
- no feeling for long-term goals;
- difficulty in designing systems and writing code for reusability;
- insufficient awareness of the product maintainability;
- no feeling at all for (or even total ignorance of) questions relating to configuration management;
- little experience of work in large, complex, and especially long projects.

In general, students finishing their studies are much better trained in computer science disciplines than in software engineering. Our experience is that the process of educating good software engineers is too cumbersome, too long, and very often unsuccessful. We claim that the students can be better prepared by having a better understanding of the software engineering disciplines if a balance between realistic examples and appropriate theory is achieved.

3 CHALLENGES IN SOFTWARE ENGINEERING EDUCATION

During the last decades there have been many discussions about education in software engineering [4][5]. Mary Shaw describes a software engineering road map [5], pointing out the main challenges that software education has to meet. In this paper we identify the challenges we have experienced when developing the course.

Challenge 1 *Striking a balance between theoretical knowledge and practical experience.*

The main challenge is to prepare students for the real world, which is inconsistent and unpredictable. The academic world is often an ”ideal” world in which students learn about problems and their solutions in a simplified form without all details. A very common solution to this problem is to execute projects in software engineering courses, based on real examples from industry [6][7]. This approach gives a more realistic working atmosphere; similar to the one the students will meet in industry. At the same time it is important that theory aspects are sufficiently discussed.

The first dilemma is how much to weight the theoretical parts in relation to the practical part. Is it better to give the

students a solid theoretical background, which they can utilize later in the "real life", or to "throw them into the water and let them learn how to swim"?

The second dilemma is about timing. Should the students first go through the main body of the theory, and then begin with project work or should they implement the theory directly as they learn it? In the first case, there is a risk that students will find the theory part too abstract and their motivation will decrease, while in the second case there is a risk that students will not acquire the holistic view of the software engineering disciplines. To avoid these risks, a balanced timing approach between theory and practice must be found, for example to gradually increase the practical portion.

Challenge 2 *Covering all disciplines vs. concentration on particular disciplines.*

Software engineering is an extremely large area covering many disciplines. Teaching at least the most important disciplines requires a complete academic program, not just a course. However, even when running "a short variant" of a program, in the form of a course for computer science students, it is essential to prepare them for the real situations in which a combination of many aspects of software engineering will be encountered. The problem is to select the most important aspects of software engineering. Even if we concentrate on the techniques and basic principles used in software development phases, the volume of the different topics the students must learn can be so great, that it is not possible to go into specific parts in any detail.

Challenge 3 *To work on 'real' projects which are large enough and small enough to be possible to complete.*

Almost every software engineering course includes elements of project work in which the students apply their newly gained knowledge. Such projects must be chosen carefully to get the right balance between implementation and paperwork. When selecting a project, it is necessary to think about the main constraint – the time frame. The course and the project must be finished by the end of the semester. This constraint is of great value, because it illustrates well what has become the most important requirement today – time to market. However, since it is not possible to compromise with time, the functional or quality requirements must be flexible and properly prioritized.

Challenge 4 *To balance teamwork with individual work.*

To prepare the students to work in large projects, they must learn how to work in teams. There are several possible strategies for grouping the students and assigning roles within each team, ranging from a purely arbitrary scheme to arranging "job interviews" to match students with their interests and capabilities. There must also be a balance between the responsibilities of the teams and the individuals. Shall the team be collectively responsible for the project or shall each individual have his/her responsibilities? Should all students work in one team, or should they be divided into several small groups? If there are several teams, should the project be organized in a manner which requires the teams to cooperate (the result from one team can be input to another), or should the teams work independently? Cooperation is frequently essential in real situations, but it can be significantly more difficult to appreciate this within a course, where the time frames are strictly defined.

Challenge 5 *To balance ideal preconditions with a chaotic environment for the students.*

One of the main goals in teaching software engineering is to simulate real situations as far as possible. As real situations are far from being ideal, one approach that has proved efficient is a "dirty-trick" model [8] in which students are confronted with unpredictable problems during the project. The supervisors play frequent "dirty tricks" on the students to simulate a volatile reality. This model is however not appropriate in a Swedish educational environment in which the relations between teachers and students are very direct, almost as between students. The "dirty trick" model would work only if the students were explicitly warned of tricks. The students would take this as a competition between the teachers and students, forgetting about the main goal of the project. Even if the students were informed about possible problems, our experience is that the students would come back and complain about the course and simply demand normal conditions should apply.

Challenge 6 *To have a proper balance between permitting the students to work independently and under a degree of control.*

One of the most important challenges is to establish good relations between students and their teachers. Teachers must be enthusiastic to make their students enthusiastic about their projects. On the other hand it is unsatisfactory if the teachers guide students too closely. In such a case there is a risk that students may stop thinking independently and begin to rely completely on the guidance of the teachers.

Challenge 7 *To achieve a good balance between long term and short term educational goals*

Faulk [12] points out the importance of having a curriculum, which combines industrial relevance with academic excellence. To achieve industrial relevance the practical parts of the course must be compatible with real industry projects, methods, tools and experiences. Lack of an industrial connection will discourage the students from taking the course. On the other hand, if a curriculum is only based on current practice in industry, the universities will not support the course, since the main goal of education is to prepare students not only for current needs but for their entire professional life.

All the challenges above must be considered when a successful software engineering course is to be set up.

4 CASE STUDY – A SOFTWARE ENGINEERING COURSE FOR COMPUTER SCIENCE STUDENTS

Course description

The software engineering course [10] was attended by students from the third or fourth year of the computer science program. The goal of the course was to give the students a general knowledge of software engineering and how to utilize this knowledge in other projects. All students were experienced in different programming and design techniques, so the main purpose of the course was not programming, but other phases of software development. To meet Challenge 1, the first half of the course was theoretical with lectures and individual exercises, concluding with a written examination. The individual exercises helped students understand the theory presented during the lectures and the examination required them to learn the theory sufficiently to be prepared for the project. The second part of the course included a practical project with the students working in teams.

Software engineering topics related to the phases of the life cycle of a software product were covered, using [1] and [2] as the main course literature. The theoretical part covered requirements, system analysis, development models, configuration management, project management, software maintenance, teamwork, etc.

To get the right equilibrium between theory and practice, ABB was invited to participate with their experience of software development. Their current project model, which is influenced by CMM [14], was used with small modifications. Following a project model used in industry, we tried to give a realistic picture of a running of a project. For example, the students calculated the project costs. Many were quite surprised by the final amount.

For the project work the focus was on teamwork, planning, process measurement and learning by doing. We divided students into teams of seven or eight students. They could choose the development model, but the main milestones were defined by us to minimize the risk of project failures. The milestones were later used to measure the accuracy of the project plan and the progress of the project. The teams reported the project status each week and each milestone had to be approved as completed.

During the project work period the students were required to analyze the problem to be solved, define and process requirements, and design and implement the system. The most difficult part was to capture the requirements, as the task they were given was to develop a program derived from a research paper [3]. The paper describes how to manage requirements with configuration management and it proposes a tool for this management. By giving the students a task related to software engineering, our intention was even more to put the focus their attention on the software engineering questions. The implementation part included partial use of existing software, partial new development. In both cases, the principle of learning by doing was applied. For development, the students had to use Visual Basic, which was unfamiliar to most. These two conditions made the students' work more difficult, but it was instructional as an example of the problems they can meet later. As students were experienced in using different languages, we expected that it would be no big problem to learn and use a new language. The most natural architecture for the problem was a two- or three-tier architecture, dividing the system into several relatively independent parts. This architecture required differentiation of responsibilities within the team, and as a consquence, intensive synchronization and integration activities. We also insisted on a strict usage of configuration management (CM). The CM-activities included version management, configurations (creating baselines) and change management [13]. A CM-tool used in ABB for many years was selected for use [9][10]. The students also used CM-related measurements to present their activities and progress.

Challenge 3 was considered when the project task was selected. To create competition between the teams, all members of the most successful team were to be awarded higher marks for the course.

We decided to have an environment to providing the students with full support. However due to a tactical mistake, the environment become the cause of many problems. We permitted, namely, the students to install the CM tool themselves, which they did with much imagination and thereby causing a number of unintentional problems. It was a good lesson for the teachers – separate development from administration.

Project efforts

During the project, the students were required to record the time spent on different activities. The chart in Figure 1 shows the average effort spent on each activity, as compared with the average planned effort.

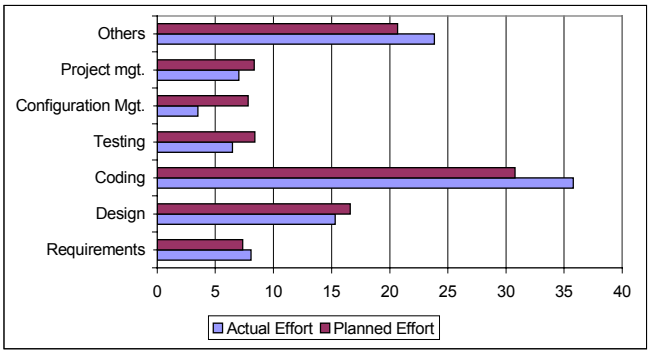

Figure 1. Planned and actual effort (%) per activity.

The first conclusion one may draw from the graph is that there is a relatively small discrepancy between the planned and actual values, which leads to the suspicion that the students did not report correctly. We have discussed this question with the students, arriving at the conclusion that students recorded correctly, in spite of a broad opinion [8] that it is impossible to measure students' efforts due to their

style of working. Due to strict milestones they simply could not afford to spend more time on a specific activity (for example design), since that would leave no time for the next activity. The graph also shows that coding was the most time consuming activity, although this was not the purpose of the course. There are several explanations why this was so. Firstly, the students had to learn Visual Basic and for many students it took a longer time than we expected. Secondly, the coding time is the easiest to measure. When we asked them why they spent less time on other activities, many of them answered that they forgot to report time they spent in discussions, small tests, and so on.

Weekly meetings were held between each project team and the teachers. At these meetings, a member of the team presented their results for the previous week and the activities planned for the following week. The students were also to report whether they felt their work was according to schedule or lagging behind. Interestingly, all the teams reported that they felt their project was on track on every meeting, although some projects were late. Over time, the students learned not to present "what is supposed to be done" but the real state of the project.

Project results

In order to be approved, the teams had to perform several elements satisfactory – starting from the analysis of the paper and the requirements elicitations, continuing with the system analysis and design, and finishing with the implementation of the application. In addition to these, factors related to the product, such as functionality, usability, and modularity, and the project activities, such as project planning, reporting, configuration management, and presentations had to be approved.

The project results varied. Two teams passed with difficulties, one of these not being able to organize their work properly. This team consisted of many strong individuals with different backgrounds, some of them good hackers, but because of weak leadership the team lagged behind the others. Although some parts of the programs were very well made, the integration was very weak. The hackers failed to inform the others of the features of their parts of the application. They included certain fancy features not specified in the requirements, but they failed to make the application sufficiently robust and general. The second team had difficulties understanding the problem – and even the aim of the course. It took them much time to get started and they remained persistently behind schedule. Two teams did a fairly good job. One team of students with less experience than average made very good progress. These students took the project work very seriously and probably gained the most in accumulating knowledge. Two teams did very well. They succeeded in implementing all the main requirements, with a good product design, good code and very good teamwork. While one team had a little better application design, the other had an excellent finalization, with very good product presentation, excellent user manual and even a printed CD. An interesting detail showed that one of these teams really enjoyed the work – it continued to work with the project even after the end of the course.

Students opinions

The students presented an evaluation of the course at its conclusion. This evaluation showed that the students appreciated working in teams and that they saw that development of software is more than coding. An interesting reflection was that the "hacker" students did not appreciate the course as much as the others.

The students found that the communication between project members and the steering group is of major importance. Some of them felt that too much freedom was left to the students, and that they sometime did not known what was required of them. This was partially true, and intended. We wanted them to be able to make their own decisions and even to judge for themselves what was required, and what they could manage. The groups also concluded that all the members in a group must have the opportunity to give his/her opinion. All the students agreed that it is of great importance to have fun during the project.

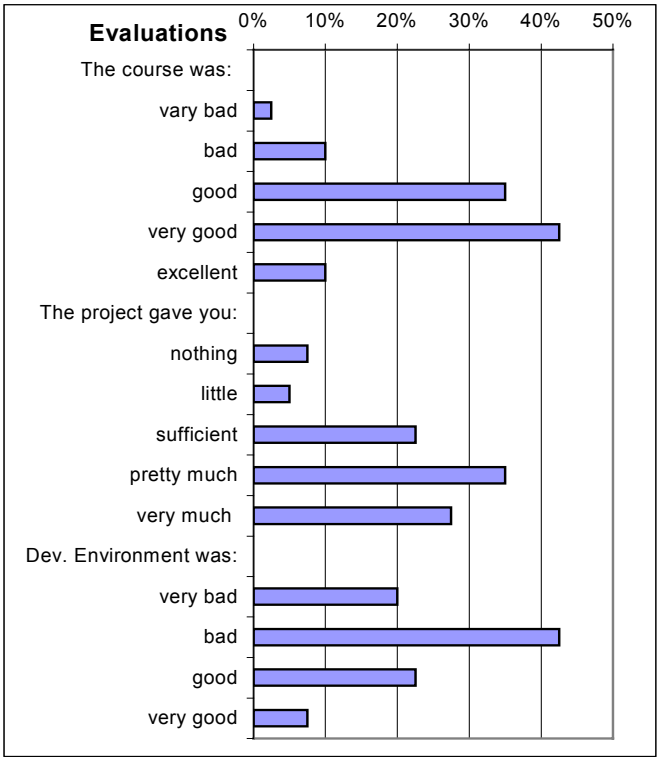

Figure 2. Students evaluation

The students' evaluation shows that they were relatively satisfied with the course, and that the project work was the more interesting part, that it gave them more. The problems they met with in the development environment (one "dirty trick") were not appreciated at all.

The complete evaluation list is placed on [10].

5 CONCLUSION

We tried to meet the challenges discussed in section 3 and to find the most appropriate balance between different requirements. We found that having the first part of the course more theoretical and the last half more practical worked well. However, the students thought it would have been better if the exercises during the theoretical part were connected more closely with the project.

Many software engineering disciplines were covered in the project, but a focus on a few more important was inevitable. Some parts, such as programming style and object-oriented programming, were not emphasized since they were covered by other curriculums. It was good to have selected topics on which we could place emphasis (project management, requirement elicitation, teamwork, configuration management). We covered other topics more briefly even if it was difficult to make the selection mentioned in section 3.

The idea of permitting the students to implement a research prototype was quite successful. Although it was not a project originated from industry, it was large and complex enough to require them to apply the knowledge obtained in the theoretical part of the course.

To provide a creative environment for both individuals and teamwork we decided to make the teams responsible for the final results. The students defined the requirements themselves, and which of these were to be implemented. They also decided which development model to use. This proved to work well for the teams in which a strong project manager was chosen, but not so for one team where the project manager was not enthusiastic enough to lead the members. This particular team almost failed to accomplish the minimum goals set by the supervisors due to stresses within the team. Other teams divided the task into smaller and more independent parts. This worked well for the most part, but one team began integrating too late and ran into problems as a result.

The incidental introduction of a "dirty trick", i.e. problems with the development environment, was very unfortunate, and even jeopardized the project success. To ensure an open dialog with the students we agreed that the supervisors could be interrupted whenever needed. E-mail and the web were also used frequently for communication. Discussions between the students and the supervisors were both at personal and professional levels. One team decided not to interact frequently with the customers (supervisors) and they did not properly understand the requirements. In general, communication between the students and teachers, as well as among the students, was very good, this being a characteristic of Swedish culture and universities' tradition. Good teamwork is also characteristic of Swedish industry.

The use of the development process model provided by ABB appeared to be very successful, although to complex in some details.

6 REFERENCES

[1] Sommerville I., *Software Engineering*, Addison-Wesely, 1999.

[2] Pfleeger S.,L., *Software Engineering: Theory and Practice*, Prentice Hall, 1999.

[3] Crnkovic I., Funk P., Larsson M., "Processing Requirements by Software Configuration Management", *Proceedings of 25th Euromicro Conference*, Milan, IEEE, 1999.

[4] Shaw M., "We Can Teach Software Better", *Computing Research News*, 4, 4 September 1992.

[5] Shaw, M., "Software Engineering Education: A Roadmap", *The Future of Software Engineering*, 22nd International Conference of Software Engineering, ACM, 2000.

[6] Dawson R.J., Newshman R.W., Kerridge R.S., "Bringing the 'Real Word' of Software Engineering to University Undergraduate Courses", *IEEE Proc. In Software Engineering*, 144, 5-6(1997), 44-48.

[7] Leventhal L.M., Mynatt B.T., "Components of typical undergraduate software enineering courses: Results from survey", *IEE Trans.Softw.Eng.*, vol SE-131,11(1987) 1193-1198.

[8] Dawson R., "Twenty Dirty Tricks to Train Software Engineers", *Proceedings of 22nd International Conference on Software Engineering*. ACM, 2000.

[9] Crnkovic I., "Experience with Change-Oriented SCM Tools", *Software Configuration Management ICSE'97 Symposium*, Springer, 1997.

[10] Software Engineering course, CD5360, Mälardalen University, http://www.idt.mdh.se/kurser/cd5360 (partially in Swedish).

[11] Crnkovic I. and Willför P., "Change Measurements in an SCM Process", *System Configuration Management Symposium*, Springer, 1998.

[12] Faulk S. "Achieving Industrial Relevance with Academic Excellence: Lessons from the Oregon Master of Software Engineering", *Proceedings of 22nd International Conference on Software Engineering*. ACM, 2000.

[13] Estublier J., "Software Configuration Management: A Roadmap", *The Future of Software Engineering*, 22nd International Conference of Software Engineering, ACM, 2000.

[14] Software Engineering Institute, "CMM for Software", SEI-93-TR-24 and -25, 1993.