A Practice Driven Approach to Software Engineering Education

Conny Johansson
Lennart Ohlsson

Abstract

This paper describes a two year undergraduate education program in software engineering. This program is designed around the principle of exploratory learning, whereby the students are trained to build knowledge by themselves and actively search for solutions to the problems they experience. In addition to the essential aspects of software engineering of managing complexity of large, changing systems and the ability to work in teams, this programs also aims at preparing the students for working in a field of rapidly changing conditions and constraints. This paper describes how these high level goals have been implemented in an actual curriculum. At the core of the program is a set of project courses which are conducted as role playing games in order to simulate the conditions in an industrial environment. Students have graduated from the program for two years now, and the paper summarizes the main lessons learned as well as a follow-up survey of experiences from some of the organizations who hired the students.

1. Introduction

Few areas of education pose the challenges software engineering does when it comes to selecting curriculum contents that will be of value throughout the students professional careers. Not only is software engineering a young area, but it is also a rapidly changing area in terms of its technological foundation: computer and information technology. This technology changes both the kind of systems to be built by software engineers and the tools available with which to build them. Progress in the tool support available to software engineers is perhaps the best illustration of the changing conditions in the field. We are in the midst of a transition from the traditional editor-compiler-debugger tool set to multiuser, repository based IPSEs (Integrated Project Support Environments) with full life cycle CASE coverage.

The set of professional abilities which is considered crucial then change as well. This makes the danger of providing skills and knowledge that soon may become obsolete an important issue in the design of a software engineering program. It is not the case to the same extent in computer science where there are many problems like language translation, search efficiency and resource management, that are more long-lived. From a software engineer's perspective however, these problems tend to move away to the domain of specialists who provide tools and abstractions on a higher level.

Software engineering also has its "essential" problems [1] like managing complexity, including communication difficulties within teams, and maintaining consistency in changing systems. Providing the abilities to deal with these problems should obviously be the core of any software engineering education program. But the "accidental" problems, which are the ones that are changing so rapidly, still put constraints when it comes to the implementation of such a program. In order to teach the essential skills we must use the conceptual and computerized tools that are available today. Furthermore, the accidental problems should not be neglected, because software engineers need the ability to solve them.

It appears as if a compromise has to be made when designing a software engineering education program. We can either de-emphasize the accidental problems or we can teach the solutions available today. In this paper we describe an attempt to resolve this dilemma by designing a practice driven education program where the students are trained to learn by themselves rather than being taught canned knowledge from books and lectures.

2. Practice Driven Learning Principles

Traditional education practices seem to be built on an assumption that the mind is a container which it is the teachers' responsibility to fill with knowledge. As pointed out by Gang [2], this view can be traced back to the era of industrialization when massive public education was born. Knowledge is dissected into separate categories and students march, as on an assembly line, from one class to another where specialized operations are performed. The responsibility to integrate the various pieces of knowledge to a meaningful whole is left to the students. Both empirical studies [3] and intuition say that such integration does not come naturally.

Learner based teaching means that education is not viewed as a process where knowledge is transferred from the teacher to the student but rather that knowledge is created within the students' minds. Courses in engineering education generally consist of both lectures and hands-on laboratory exercises. The course content is traditionally defined by the lecture series with the exercises giving additional support. The underlying model is that theory should be first taught and then verified in practical exercises. We have adopted a more practice driven education where theory is regarded as something which cannot be taught but must be built by each individual. In our approach we therefore have the hands-on exercises to define the course contents and regard the lectures as supporting activities. This framework, that goes from the concrete to the abstract, capitalizes on the innate human desire to explore and learn and encourages learning that is characterized by "practice-pull" rather than "theory-push".

It should be noted that the emphasized practical work is not meant to be at the expense of abstract and theoretical reasoning. Such level of knowledge is a superior form of knowledge, but only if the student has learned it deep enough to be able to apply it on non-schoolbook problems. By obtaining abstract knowledge through guided reflections

on one's own experiences this knowledge will be acquired on a higher level of understanding than can be done from a book.

Working with the premise that a person can learn only by himself has quite an effect on the role of the teacher. It requires taking a very humble attitude toward teaching. Rather than just applying an analytic knife and cutting the area of study into pieces small enough for the students to swallow, the main task of the teacher is to encourage students to take initiatives on their own. Taking a seemingly more passive role in class does not, however, reduce the teacher's total effort. On the contrary, maintaining educational progress in this role in class requires additional effort to be spent on preparation.

A prerequisite for a more learner-centered education requires a change in the currently distributed responsibility for engineering curricula over a large number of departments. A teacher must not only know the subject he is teaching but also how this fits into the larger picture. In a situation of choice, a student is aided more in his learning process by a teacher who has an adequate knowledge of the context of his area than very specialized knowledge of his particular domain of expertise.

3. Curriculum Overview

Our program is a two year program leading to a University Certificate degree which is offered by Swedish universities as a complement to the four and a half year degree Masters programs offered at the traditional engineering schools. The students thus have no preparatory computer science or other academic training when they enter this program.

The principle of practice driven education is applied both on the level of individual courses and on the curriculum level. The large proportion of the curriculum consists of project courses (40%), and most of the other courses are designed around laboratory exercises.

To promote and utilize the potential of the exploration which learner do spontaneously, it is also crucial to have a comprehensive work environment. From the first course, the students have access to Unix workstations, and throughout the courses they are successively introduced to a number of modern tools.

Given below is a summary of all the courses in the program with an emphasis on how our pedagogic foundation has been implemented for the subject area of software engineering.

1st Semester

Computer Science
An important criteria in the choice of the primary programming language in a software engineering education is, that it is or is likely to become widely accepted in industry. On

the other hand, the language should provide reasonable support for modern programming practices and be compatible with tools like database management systems, user interface management systems, etc. C++ is the language which currently best fits these criteria. By using C++ in this five week introductory programming course we have been able to introduce the object-oriented philosophy as the first way the students learn to think about programming.

The teaching of the programming language is inspired by the way humans learn natural languages, by absorption and imitation. Before the students are asked to express something that requires a new language feature, they will generally have read it several times. From the beginning the students are presented with a fairly large program where difficult parts initially are hidden by abstraction barriers. By doing carefully scoped exercises that modify this example in various ways, the students' active vocabulary gradually is extended to cover the whole language. Embedded in the example are also instances of standard algorithms and data structures which the students then encounter during the exercises. The method resembles the one used in Abelson et al. [4]. Apart from the improved motivation that comes from working with more realistic examples, this method has the benefit that the abstraction principle is learned not as yet another technique but as a natural and obvious way to manage complexity. Always programming in a context of other modules also promotes a habit of using, and later re-using, existing software.

Written and Oral Presentation.
Presentation skills form the basis for producing quality documentation of software systems as well as working together in a team. The written word is the vehicle for collective thought and a means to assure that ground covered in a project is not lost. This is particularly important for a software engineer since software essentially is various forms of descriptions of a system, and the basic principles for making understandable presentations are the same as those for making understandable software.

In this course the students give a number of mini-lectures and hand in some written assignments on which they receive individual feedback on disposition, style and overall clarity. The main written assignment consists of writing a beginners manual to the Unix system, which also serves as a vehicle for the students to learn the environment by themselves.

Analysis and Design Methods
Like the computer science course, this course also uses a large example. Again, the size is used to motivate the use of methods and notations for analysis and design. The students are asked to do a modification of such complexity that it is very hard to do on the code level directly. With a system that is documented in a comprehensive analysis and design notation, in our case OMT [5], the task is manageable. Through further exercises, the students progress from understanding the notation to using it on systems they design themselves.

Digital Design and Computer Organization
In this course, which covers the low-level end of the software engineering spectrum, the

students work through two lab series. In the first series they use Boolean algebra and finite state machines to construct combinatorial and sequential digital circuits, ending by building a simple CPU. In the second series they work in assembly language on an HC11 microprocessor system to explore the principles of computer organization. In the final laboratory the students build a floppy disc controller, which is later used in the operating systems course.

2nd Semester

Discrete Mathematics
The purpose of this course is to give the mathematical basis of programming. Although it is the most theoretical course in the program it is still based on laboratories and practical exercises. By using a mathematical software package, Mathematica, and conventional programming exercises, the students solve problems in set theory, graph theory, combinatorics, matrix theory and do simulations of finite state machines and Turing machines. Basic concepts like sets, tuples, mappings, trees and graphs are defined and applied as abstract data types in practical programming exercises on concrete problems. Possible trade-offs in the implementations are studied. This serves as motivation for learning complexity theory as a tool to make such trade-offs correctly.

Operating Systems and Real Time Systems
This course contains three series of laboratory exercises. The first part uses the floppy disc controller built in the course on Computer Organization and extends it to become a small file system. The second series uses Ada tasking, both to illustrate high level constructs for concurrency management and real-time programming and to build a simulation model of a virtual memory system. This model is used to study concepts like thrashing, swapping and page replacement strategies. The third part studies file and process management in Unix including X11 window system.

Individual project
This course is the first of three project courses in the program. To give some realism to these courses they are run as role playing projects, similar to the approach described in Jacquot et al. [6]. These courses also try to introduce a professional attitude. A professional is distinguished by an ability to make assertions regarding the cost and time required to deliver a product or service and to maintain a responsibility to fulfil this promise. A professional is also able to deal with those unfortunate, but inevitable, situations when he is unable to fulfil his commitment. In those situations he takes responsibility not only to minimize his own losses, but also to initiate negotiations to find a solution which, under the new circumstances, minimizes the consequences for the client. This behaviour is the natural result of an active commitment as opposed to being more passively assigned a task, as is the common situation in education environments. We have introduced the concept of 'commitment culture' for this style of working where commitments, or active and voluntary responsibility, are used as a driving force in everyday work [7].

This course runs full time during five weeks and the size of the programs delivered is approximately 5000 lines of code. The students are shown a number of existing applications and asked to do "one of those". Stating the assignments in this way naturally motivates the students to spend substantial effort on early activities like requirements analysis. The applications range from games to simple spreadsheets to process kernels for embedded systems. In the first course, the customer is only the user of the program. The required delivery only includes an executing program and user documentation, unless additional deliverables are agreed upon in negotiations.

The student's first task is to make a specification of the application which can form the basis of an agreement with the customer and to negotiate this with the customer. The negotiations not only involve specification of the product, but also the acceptance procedure in order to minimize the risk of deadline overrun, i.e. not passing the course. Things like early delivery and acceptance of a specification document in a graphical notation or of a functionally incomplete (but debugged version) of the program, may be agreed upon as part of this procedure.

The voluntary aspect of a commitment is emphasized throughout the course. The students are encouraged to re-negotiate their tasks by the customers/teachers. In this first course the students are rather complaisant negotiators, but the breaking of an agreement is however never accepted. Not even absence with illness is accepted if the customer has not been promptly informed.

In addition to teachers playing customers, there is also the role of technical advisor. No teacher plays the role of both customer and technical advisor to the same student in order to make roles as distinct as possible.

3rd Semester

Basic tools and techniques

There are several basic tools and techniques that are part of a modern software engineer's toolbox. With our condensed curriculum we cover four such areas in one course:

- Database design
- Human computer interaction
- Translation tools and techniques
- Data communication and distributed systems

The database part starts by introducing a modern, powerful development environment (INGRES) for these kinds of systems. It is introduced by presenting a complete and running application. The students learn the environment by modifying the example in various ways. This is supported by lectures on conceptual modelling and relational database design theory. The final exercise involves the modelling of a rather large organization and designing and implementing an information system for it emphasizing documentation and performance optimization.

The human-computer interaction part of the course partly uses the same examples as the database part and partly uses a stand-alone user interface management system (GUIDE). Ergonomic rules for the design of menus, forms and reports are studied, and existing style guides are compared with the user interfaces of various commercial products.

The part on translation tools and techniques starts by using the pattern-scanning language awk to do simple reformatting and translations of files. Its limitations then motivate the introduction of the more powerful tools lex and yacc. Different classes of grammars are studied and the tools are used to build a simple command tool.

The part on communication and distributed systems is done in the framework of the layered ISO/OSI model. We selected two levels for the practical exercises. First, a file transfer protocol between a Unix-workstation and a micro-controller is implemented. Second, a client-server application is built on the RPC-mechanism and its associated development tools.

Group project

In this course the students work in teams of four to five during ten weeks. The course consists of two parts: the first part is the development of a new system, and in the second part the teams are asked to make certain modifications to those systems. The parts are both run as complete projects with development task approximately twice the size of modification task. Examples of systems built in this course are:

- An online measurement data collection system including sensor interfaces on single board computer and transfer to a database on a workstation for presentation to the user.
- A host/target development system for the HC11, including an assembly level debugger on a workstation.

The goal of this course is to introduce informally project planning as a refinement to making commitments. The small team size in this course makes it possible to work in an intuitive, informal organization. Just working together, however, always poses problems which the students have the opportunity to solve on their own. This creates the appreciation basis for a more strict organization which is introduced in the final project course.

The project management in this course practices tracking and successive planning. Achieving quality, i.e. customer satisfaction, must be addressed throughout the project lifetime. The key is to achieve mutual commitment, to have the customer involved in order to get continuous feedback and, hopefully, a blessing of the approach taken. Minuted meetings with the customer are used to formally agree progress, and the issue of quality control is transformed to the problem of having sufficient information at these meetings in order that the customer can make competent decisions.

In this course the teacher's role of technical consultant is augmented with a function of a quality controller. His task is to assist the teams to organize their work internally, but his

role is only to give advice and let the students decide for themselves how they want to work. He participates by being present at the groups' meeting once a week.

Including a modification task in the course motivates development in such a way that it facilitates future unknown changes.The problems of maintenance are made realistic and very concrete by having the groups switch systems so that everybody is confronted with code and documentation developed by someone else. The need for different kinds of system documentation is thereby further illustrated.

Project organization and human work science
How to work together in a team is not explicitly taught in the Group Project course. Instead this non-technical course is given in parallel. It highlights topics like planning, distribution of work, leadership, conflict management, compromises and communication difficulties. The concrete experiences from the Group Project course provide motivation and reasons to actively discuss and find solutions to these issues. This course also covers the concept of commitment in more detail and its basis in that everybody must take active and voluntary responsibility to find and give information and to maintain an open and communicative atmosphere.

4th Semester

Large project
In this course, which runs for 15 weeks, the students work together in teams of 12-15. The first year this course was run meant that all student was on the same team. The project is divided in a pre-project phase and a development phase. During the pre-project phase, which constitutes about one third of the course, the problem is defined and the requirements are determined.

The development model used is based on evolutionary delivery [8], which has the following characteristics:

- Planning for multiple objectives, not only what will be done, but how well.
- Early and frequent iteration, early feedback with the user by early delivery.
- Completion of analysis, design, build and test in one step.
- User oriented approach, the requirements are redefined after each delivery.
- Open-ended systems architecture with strong aspects on non-functional requirements.

At this point in the education, the students have gained an appreciation for quality management. In the pre-project phase they develop a quality plan for the project, starting from a quality standard in accordance to the principles of ISO9000 [9]. The quality plan defines procedures for configuration management, quality assurance and the various kinds of documentation. Quality assurance is primarily based on inspection techniques.

The project leader in this course is an external software consultant. As before, the

customer roles are played by teachers. Other roles in the organization defined in the quality plan like subproject leader, configuration manager, delivery manager, documentation manager, inspection leader, etc., are all played by students.

4. Experiences

In the practice driven courses it turned out the amount of work the students had to put in to complete their hands-on assignment was noticeably increased compared to conventional teaching. During this time the students had no additional teacher guidance. We found that giving the students the freedom to make and learn from their own mistakes provided them with sufficient motivation to make this extra effort.

The students were enthusiastic about the role playing approach used in the project courses. It turned out that as early as in the individual project, the students needed very little technical guidance and support. They were able to find answers to their problems in manuals, literature or sometimes from the tool suppliers. We learned not to underestimate the technical abilities of the students. On the other hand, the students found out that the deceivingly simple problem of making and keeping promises hides many difficulties to be mastered. Negotiating with a customer was a new experience to most students and in the first attempts they generally accepted too much work. They quickly learned the rules of the game, however, and soon started to explore the limits of their power.

We learned that it was vital that the teachers play their roles distinctly and clearly demonstrate which "hat they have on". The teachers often knew the technical problems "too well" and had to be careful not to include any advice on the solution when stating their needs as a customer.

The lack of project management lectures in the group project course created some initial difficulties for the students in dividing the work among themselves and at the same time maintaining sufficient communication about the various parts of the system. From the beginning, though, they acted professionally towards the customer and concealed all internal conflicts.

Running the course in Project Organization and Human Work Science in parallel with the Group Project turned out to be successful with mutual benefits to both courses. Recently having had concrete experiences, the students were strongly motivated for discussing problems of human interaction.

The idea of rotating the groups for the modification part of the Group Project turned out very well. Although sometimes quite severe criticism was raised about documentation quality, code structure and design choices, the discussions were constructive without any bad feelings.

Grading project courses is a difficult issue. We grade a group as a unit as either pass or fail, where the grading criteria is the performance from the customers point of view.

Failures on a project course are very rare since the customers have the right to claim a compensation in the form of extra functionality when the final result can not be accepted. When this "fine" has been fully paid, the group has passed the course.

The large project course this first year was very much a learning experience for the teachers as well as the students. We learned, for example, that we need to improve the project management. Having an external professional as project manager was very valuable, but his limited presence at the university created some difficult communication problems. Next year, we intend to separate the responsibility into two roles, with an external person still playing the role of the official project manager but with a permanent teacher as an assistant project manager. To our satisfaction, we noted that the students eagerly accepted the need for formally defining the development process and its quality procedures.

When the large project started, the problem was less well defined than we had wanted. Although this caused quite a bit of confusion and inefficiency, it also gave a realistic illustration of real life development, where the developer must take active responsibility to ensure he is solving the right problem.

A serious issue is the grading of the project courses. So far we have graded a group as a unit, either pass or fail, and the main grading criteria has been the performance from the customer's point of view. Whether this is the right way to do it, we do not yet know.

Almost a year after the first students graduated we did a follow-up with their employers. The general comment was that our students needed less time of internal training to become productive compared to students who had gone to ordinary engineering schools for four and a half years. One employer explicitly said he preferred hiring people with this background. In particular, their project experience was highly valued. As new engineers they had an appreciation for the different roles that are needed in a development, so they could easily start taking responsibility as configuration manager, documentation manager or basic test manager. No employer could mention an important area in which the students had insufficiently training.

Being a teacher in practice driven education is both rewarding and demanding. It has given us great satisfaction to work with students that are active and eager to search for knowledge. It is also difficult because we are influenced by the conventional style of the teachers during our own education. Using a practice driven approach, the teachers must live as they preach and actively search new knowledge by themselves.

References

[1] F. P. Brooks, "No Silver Bullet - Essence and Accidents of Software Engineering", IEEE Computer, Vol. 20, No. 4, pp 10-19, April 1987.

[2] P. S. Gang, "Rethinking Education", p70, Dagaz Press, Atlanta, Georgia, 1989.

[3] J. M. Carroll, "The NÜrnberg Funnel", The MIT Press, Cambridge, Massachusetts, 1990.

[4] H. Abelson, G. J. Sussman, J. Sussman, "Structure and Interpretation of Computer Programs", The MIT Press, Cambridge, Massachusetts, 1985.

[5] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, W. Lorensen, "Object-Oriented Modelling and Design", Prentice-Hall, 1991.

[6] J. P. Jacquot, J. Guyard, L. Boidot, "Modelling Teamwork in an Academic Environment", Proceedings of the 4th SEI Conference on Software Engineering Education, pp 110-122.

[7] L. Ohlsson, C. Johansson, "An Attempt to Teach Professionalism in Engineering Education", World Conference On Engineering Education, Vol. 2 pp 319-324, Portsmouth, September 1992.

[8] T. Gilb, "Principles of Software Engineering Management", Addison-Wesley, 1988.

[9] ISO, "ISO 9000 - Quality management and quality assurance standards", International Organization of Standardization, 1987.

Footnotes

The authors are with the Department of Computer Science and Business Administration, University of Karlskrona-Ronneby, S-372 25 Ronneby, Sweden