Early Practise and Integration - The Key to Teaching Difficult Subjects
International Symposium on Software Engineering in Universities, ISSEU ´97, Rovaniemi, 1997, pp 93-100

Early Practise and Integration - The Key to Teaching Difficult Subjects

C Johansson
Ericsson Software Technology AB,
University of Karlskrona/Ronneby
Soft Centre, S-372 25 Ronneby, SWEDEN
E-mail: Conny.Johansson@ide.hk-r.se

Abstract

Software Engineering is a young area compared to other engineering disciplines. The subject is rapidly moving and more questions than answers seem to appear. Usually, Software Engineering projects involve several people for a prolonged time. Large projects range over several months or years and involve considerable groups of people developing massive systems. Such systems are complex because of their size, and because they need to be regularly modified in order to meet new and changed customer requirements, and because of the need to correct faulty products.

We have found it very important to highlight as many questions as possible that appear within Software Engineering already in the education programme. The strategy is to provide as much practical experience from projects as possible, and to let the students be aware of questions and problems before they get possible answers and solutions to those questions. In order to have the right practical training, you have to initiate active tasks on the courses. Some parts come automatically from initiatives from students, but some parts need to be coerced by the teachers. We have noticed that it is important to introduce specific fields in Software Engineering, with a successively increasing emphasis. To achieve high understanding, the order and emphasis need to be carefully considered. This paper describes five fields that we think should be emphasized in this way. These fields are specification, system decomposition, planning, tracking and verification.

1 The Software Engineering Programme

The University of Karlskrona/Ronneby has seven years experience of running a Software Engineering programme, which has a strong emphasis on practical sections such as project courses. The technical content, as well as the educational approach, changes and improves continuously as we gain more experience from course evaluations and influence from both industry and current research. The programme covers four years and the number of project courses, where projects are executed just as real projects should be, comprises of 25 percent of the total study period. As the basis for teaching professionalism we use the concept of

Year 1

Object-oriented Programming in Java
Discrete Mathematics
Data Structures and Algorithms in Java
Automata and Formal Languages
Object-oriented System Development
Digital Design and Computer Architecture
Database Systems
Individual Software Engineering Project

Year 2

Real Time Systems
Computer Networks
Operating Systems and Distributed Systems
Compiler Design and Translation Techniques
Human Computer Interaction
Project Organization and Human Work Science
Small Team Software Engineering Project

Year 3

Software Maintenance
Logic
Software Architecture
Business Administration
Large Software Engineering Project

Year 4

Project and Quality Management
Requirements Engineering
Advanced Object-oriented Concepts
Software Metrics
Software Economics
Verification and Validation
Master's Theses in Software Engineering

Figure 1: Recommended course curriculum for the Software Engineering Programme

commitment culture, a working atmosphere built on voluntarily taking responsibility. In the first project course, which takes place towards the end of the first year, the students do small individual tasks lasting for five weeks. In the following three project courses the team size and the project size are then gradually increased. In the second project course the team consists of four to five students and the project lasts for ten weeks. The third project course has the same team size as the second one, but lasts for five weeks and comprises Software maintenance. In the final project course the students work in teams of twelve to twenty on a twenty-week project, which is about 10000 man-hours. A summary of the programme is presented in Figure 1, and the courses in more detail in WWW [1]. See Ohlsson [2] for information of how project courses are executed, and Johansson [3] for experiences from using industrial client's projects. In addition to the recommended course curriculum, we also have themes for each year: Written and oral presentation, consultancy and research methodology. These themes are executed as a part of project courses and Master's Theses, but also integrated into the other courses in the curriculum.

2 Five Important Fields in Software Engineering

Most Software Engineering education rely heavily on the foundations of Computer Science and Mathematics. A first-class programme should, however, not only comprise of courses within these two subjects. In order to become a professional Software Engineer you need to understand and have knowledge about subjects such as Industrial Organization, Business Administration and

Human Work Science. It is important to have separate courses in these subjects, but it is even more important that these subjects are integrated into the 'technical' courses. This is very difficult in a traditional academic environment where each institution and department defend their subjects, but it is very important for the students so that they can improve their understanding of how the different subjects fit into their profession.

We have identified five important fields, which are more related to other subjects than Computer Science and Mathematics. These fields are Specification, System Decomposition, Planning, Tracking and Verification. They are hard to get the hang of, and need to be integrated into as many courses as possible. In order to achieve an acceptable level of understanding we introduce them early in the course curriculum, and increase the amount and complexity gradually as the students gain more understanding and knowledge within the fields. These fields are traditionally not a part of Computer Science course, but should absolutely be included in a Software Engineering curriculum.

3 Specification

It is difficult to elicit and capture customer requirements and even more difficult to express requirements in a written customer requirements specification. Written requirements need to be interpreted identically by all the parties involved. Written requirements specifications are introduced during the first year in the Individual Project course. We do not, however, have emphasize on writing a 'correct' specifications from the very beginning. The assignment is to sign a contract that specifies the agreement with the customer. Awareness of the importance of the requirements specification increases during the project and when the project is concluded. Ambiguity, interpretation and verification problems when dealing with requirement specifications need to be negotiated at an early stage.

The customer requirements are often insufficiently detailed. It is often necessary to create an internal project requirement specification. This System specification includes additional requirements that the supplying part sets on the project. Additional performance, quality and system requirements may be included in this specification. Here you can use more formal specification methods, i. e. Z, which seldom can be used in the customer requirements specification. More methodical specification techniques are studied later on in the 4th year.

The specification problem is, however, not only a problem that relates to the customer requirements specification and the internal System specification. Large systems need more man-hours to develop and more people need to be involved in order to achieve an acceptable lead time. The system needs to be divided into manageable parts and the project into sub-projects. The solution is sub-system

specifications, interface specifications between sub-systems and test specifications on all levels.

The biggest difficulty is to teach the students the importance of specifications on sub-system levels. The students can not achieve this before they have participated in a project which is large enough to be divided into several obvious parts. We find it hard to motivate the students to create these specifications. Extra support and teaching is needed in order to get anything done in this area.

It is important that students have a lot of experience, we recommend at least one year, from developing products, before you teach more methodical specification techniques. It is practically impossible to try to teach this earlier because of inadequate understanding of the problems that the methods are supposed to solve.

4 System Decomposition

Systems, which take about 1500-2000 man-hours to develop, may be developed by small teams which are not organized in a formal way. In these teams, i. e. 4-5 people, it is common that team members knows ‘all’ about the product’s properties, which is possible just because of the small size of the product. We make, however, the student aware of decomposition by forcing them to do time estimates on work packages at a suitable granularity, i. e. work packages not greater than 20 hours. This is done as early as in the 1st semester! The verification that students learn something from this is given later on, i.e. 2nd semester, when most of the students divide the system into smaller parts without being forced to. The system decomposition field is tightly related to the specification field - there exist requirements for each component.

We have noticed that system decomposition is very difficult for the students to grasp. It needs to be systematically introduced and the students not only need experience from developing products, but also time to reflect on these experiences. Therefore, we have reinforced the Software Architecture part of the curriculum with a separate course, which is given between a small project course and a large project course. Our aim in this course is to introduce more advanced architectural models and methods, and to reflect on the product structure and design they have developed in previous projects.

We have noticed that it is important to emphasize the issue of responsibility. If responsibilities for sub-systems and work packages are appointed, there is a greater likelihood that the specifications are clear, the design is good, and that the progress of the work packages is better monitored and thus makes better progress. It is also of great importance that the teachers put extra effort into support when reviewing system design.

5 Planning

Deficiencies in project planning seem to be common in many companies despite the working field. Planning Software projects is probably not more difficult than planning the building of a house or a motorway, but lack of historical data makes Software project planning difficult. In all projects the primary factors are size, cost (including resources) and time. Estimations should be based on experience and historical data which should be available in the company you are working at.

In order to create the historical database, we include time estimates from the very first course. Without teaching how estimations should be done, we start by compelling students to do them. We are not concerned with the estimated size, they vary a lot, but we do get the students into the habit of using estimates. In the subsequent course we force them to do smaller granularity of estimations, e. g. each part of the estimation shall not be greater than 10 hours, and code size estimates. This forces the students to divide the system into smaller parts, and to think in terms of the size of the product. The following step is to review the estimations continuously while the project progresses. When necessary, we re-plan and re-estimate if the estimations differ too much from the initial ones.

It is important that the estimations include more than time estimates, but also size (number of lines of code, number of pages, number of classes, etc.), and number of resources. The size metrics should be of value for the project, more alternatives to lines of code is of course possible. The cost metrics should not be forgotten even though we are running projects at University level. By just talking about costs we get students used to them, and acclimatize them to understanding how managers, which are responsible for budget, are reasoning.

The motivation for dividing estimations into smaller parts was rather difficult at the beginning. Students in previous study years helped us, however, with this dilemma. We have noticed that students rely and trust experienced students more than teachers in this case (as in many other cases). And if older students say that it is obvious that you should do estimations, the inexperienced students then have enough incentive to do it.

As a teacher, it is important not only to check the plans and performance on a team level, but also to scale the process down to a personal level. Guidance, support and feedback on a personal level must be given. But remember that you cannot cover everything as a teacher - You have to be selective and choose some individual reports and discuss the common difficulties. Examples of such difficulties are that some activities are not planned, difficulties in reporting progress and mismatch in numbers in individual reports and in the team report.

6 Tracking

It is important to maintain an updated project plan which reflects the current

status in the project. An outdated project plan is of no value. We do not introduce tracking until the first project course at the end of the 2nd semester. With the help of a meeting half-way through the project, we enforce simple tracking. The students shall be able to answer questions such as: "How many hours have you worked?", "Is this according to plan?", "Are the size estimates still valid?" and "How many hours have you left to work before the product is ready for delivery?". These are examples of questions that any person who is responsible for the project budget should ask. We have found that it is rather easy for the students to realize the importance of progress tracking if you handle it in this way.

When the students start working in teams we introduce the earned value method, see for example Boehm [4]. Using this method with visualization in simple diagrams for planned/actual/forecast progress and cost, great value is added to simple progress reporting, and it is rather easily understood by the students.

The most difficult task to do in this matter is to estimate the real progress by the "percentage done" principle. The students need guidance from teachers in system decomposition and activity division. Once you have divided into activities and tracked to see if the activities have been completed, one can easily put this in proportion to the total amount of estimated time.

Another important matter related to project planning and tracking is to define a clear criteria for re-planning. The project plan should be updated if, for example, one of the important intermediate milestones in the project is delayed. Without having a clear criteria for re-planning, you will most probably have an outdated project plan in the project.

7 Verification

The most common verification activity used, is testing. In order to emphasize the importance of delivering flawless products, we get the student used to testing activities by mentioning it from the very beginning. In the first course, in a small laboratory task which comprises of about 40 hours, we require the student to report the number of faults found in tests, and let themselves define what they mean by testing. We get a great variety of numbers, all numbers between 0 and 500 seem to be represented, but the students become aware that testing should be a part of the development cycle. In the subsequent course the students have to specify the most common types of errors, and make suggestions about how to avoid these errors in the future. The next step is to test another students implementation. This is in order to accustom students to do independent testing.

Systematic tests are introduced in the Small Team Project course. The verification focus is System Test planning, execution and reporting. The planning phase includes, i. e. preparations in order to execute the tests, test case

specification and criteria for passing each test case. The students must formalize this and write a System Test Specification as soon as the Requirements Specification is approved. Systematic regression tests are handled in the Software Maintenance course.

In the Large Team Project course, we have chosen to focus on System and Integration Test. We are not so concerned about the tests on lower levels, i.e. functional unit tests and basic algorithm tests, if the students do not want to do them themselves. Code reviews are also included in this course, but they seem to get neglected if the timetable is threatened in any way. It is better to focus on a few important test activities than everyone that indeed needs to be executed. In this course we also do root cause analysis on the faults. This includes i.e. analysis of what type of fault it is, where it should have been detected and what process need to be improved in order to avoid this fault from occurring again. While this project is divided into at least two deliveries, it is possible to make use of these analyses directly in the on-going project.

We have found it important to focus on few test activities. Even though we do not reject formal lower level tests, we find it more important to perform System and Integration tests more thoroughly. The students have enough problems in specifying test preparations, test cases and criteria for passing the tests and therefore, it is in these areas that the focus and support should be concentrated. In the final year the students are prepared for studying methods for verification and validation. Reliability and performance testing are important parts of the course this year.

Verification does not, however, include only traditional error detection methods like testing. Reviews and inspections have more focus on early detection. Although these kinds of techniques have been known for two decades they are not so widely used in industry today, and are almost non-existent in academic education. Informal reviews are introduced at the beginning of the 2nd semester. As design is one of the most important work results to review, we start by letting the students review each others design. Written reviews are handed over to the authors of the design. In the subsequent course we extend the review concept by also reviewing contract and requirements specifications.

Furthermore, in the project courses we first let everyone in the team participate in informal reviews of requirement and test specifications, project plan and design. Formal inspections are practised in the large team course. Note, however, that we do not force them to use any specific inspection method. The students create their own method based on their experiences from informal reviews, by studying some references, see for example Gilb [5].

Just as it is in many commercial projects, introducing inspections to an organisation is extremely difficult to do. Time pressure, not enough time for preparation, ineffective meetings, etc. are common problems when using

inspections. By letting the students experiment with inspections rather freely, we prepare them better for deeper studies in different methods for performing inspections. In the 4th year there are specific parts which include inspection technique studies. One of the most interesting sessions is the argumentation sitting, where students are divided into two different teams, when one team is arguing FOR inspections and one team AGAINST!

8 Conclusion

The educational model we use gives the correct Software Engineering profile that an education in this area should have according to our experience. The students have a clear identity, but also great potential for specializing in any field within the subject of Software Engineering. However the demands on the teachers qualifications and skills are high. You must as a teacher cope with all the wicked situations where you do not really know the correct answer. But by having an open-minded atmosphere with free discussions about results, problems and possible solutions, you will have a better chance of success.

As the years pass, we have found that our role as teachers has become easier. The students communicate problems and solutions among each other and try to solve it in this way first. This is a creative environment and according to us produces a very good atmosphere.

Some of the solutions described in this paper seem to imply a great deal of bureaucracy and that a lot of documentation needs to be produced. If you emphasize the importance of everything being documented, and not necessarily in separate documents, you will decrease the supposed bureaucracy. However many students seems to advocate more bureaucracy than the teachers request!

Students, who have graduated from our programme, are better prepared for participating in real projects. There is little difference between problems that appear in commercial projects vs educational projects. Students can quickly be ordinary team members, and commit to roles such as customer service, project managers, designers, estimators, inspectors and testers.

9 References

[1] WWW URL: http://www.ide.hk-r.se/education/pt/english.

[2] Ohlsson L, Johansson C, ”An Attempt to Teach Professionalism in Engineering Education”, 3rd World Conference on Engineering Education 1992, Vol. 2, pp 319-324.

[3] Johansson C, Molin P, ”Maturity, Motivation and Effective Learning in Projects - Benefits from using Industrial Clients”, Software Engineering Education 1995, Vol. 2, pp 99-106.

[4] Boehm Barry W, Software Engineering Economics, Prentice Hall, 1981.

[5] Gilb T, Graham D, Software Inspection, Addison-Wesley, 1993.