Experiences from a Project Oriented Software Engineering Course

Jürgen Börstler
Department of Computing Science
Umeå University, Sweden

UMINF-95.27
ISSN-0348-0542

Abstract
In this experience report we describe early experiences with a new software engineering course held at Umeå University, Sweden. The course is made up of lectures, formal meetings, presentations, and a project part. Project proposal, requirements engineering, prototype implementation, and prototype evaluation for each project is done by different teams. A tight schedule of deliverables and presentations give the lecturer some means for project control.

For analysis, design, and implementation the course focuses on object-oriented approaches. Tool support is provided for most of the tasks.

Two things have shown up to be critical in this type of course: The selection of the “right” projects and the “quality” of the design documents. For some types of projects only a small amount of the techniques taught in the lectures is needed. It is therefore important to select projects where the student can apply all aspects of object-oriented modeling. To produce understandable and implementable designs is very important, since the design document is the basis for subcontracting another team for part of the implementation.

Key words: Course Design, Student Teams, Projects, Tools, Deliverables.
1 Introduction

Software engineering is a discipline which evolved to cope with the complexity of large software systems and their production. A deeper understanding of this discipline can only be gained by practical experiences ([MoPo 93], [Myn 90]).

Most of the subjects in a typical software engineering curriculum are directly or indirectly related to size, complexity, and quality of software. Approaches to deal with size and complexity are difficult to teach in classroom lessons alone. In the lecture most of these approaches seem to be trivial, since there is not enough time to introduce complex examples. Small exercises do not help either, since the real value of software engineering does not proof itself to be useful for them. The application of software engineering approaches rather seem to cause a lot of superfluous documents to be produced.

In software engineering education it is therefore necessary to let the students solve a problem which fulfils the following requirements:

- It is to big or complex to be solved by one or two students in the given time.
- There is no guarantee that all the inputs are complete and consistent.
- There are people with different kinds of knowledge and backgrounds involved.

2 Course Outline

The software engineering course at Umeå University actually consists of a practical and a theoretical part, which run in parallel streams. The theoretical part is a series of lectures to present software engineering knowledge. In the practical part the students can immediately apply their knowledge to a project. It is a third year course, i.e. the students already have some practice concerning small programming projects.

The lectures cover most of the typical software engineering topics with an emphasis on object-oriented analysis and design (see figure 1 for more details). The texts of Pressman (/Pre 92/) and Booch (/Boo 94/) are used as references for software engineering in general and object-oriented analysis and design in particular.

For the project part the students form team of 3–6 students. The project part is strongly oriented towards the the deliverables the teams have to produce.

Each team is required to hold regular formal meetings to support communication between team members. The minutes of these meetings are part of the final report. Each team is involved in several projects since:

- A team must not select its own proposal as its project.

Students do often have big problems writing down requirements for their own ideas, since in their heads most of the things are “crystal-clear.” They therefore see no need to write all these things down and create uncomplete requirements documents which are difficult to understand for others.
• Each team must subcontract another team for a part of the prototype implementation.

  This confronts the teams with the importance of fixed and clear interfaces. They cannot change the design at will during implementation, since other teams are affected.

• Each team must evaluate another teams prototype.

  This was introduced to motivate the production of a user manual and to assist project evaluation.

Each team has to assign specialist roles to its members. Each role comes together with responsibilities for one or more deliverables and/or presentations. This gives the lecturer some means to control that all students in a team do some work.

The following table gives an overview over the contents and schedule of the course.

<table>
<thead>
<tr>
<th>week</th>
<th>lectures</th>
<th>deliverables</th>
<th>presentations</th>
</tr>
</thead>
</table>
| 1    | Introduction to SE  
      | Project Planning  
      | Requirements Engineering  
      | Team description  
      | Project proposal |
| 2    | Classical Requirements Engineering (ER, SA, SA/RT)  
      | User Interface Design |
| 3    | Object-Oriented Analysis  
      | An Example using S-CASE® |
| 4    | Classical Design  
      | Object-Oriented Design  
      | The example continued |
| 5    | Detailed Design  
      | Formal Specification  
      | Prototyping  |
| 6    | Object-Oriented Programming  
      | Coding Guidelines |
| 7    | Testing |
| 8    | Configuration Management  
      | Cost Estimation  |
| 9    | Test Cases |
| 10   | User Manuals  
      | Prototype Demo |
| 11   | |
| 12   | Final Report  
      | Prototype Evaluation |

Table 1: Contents and Schedule
3 Usage of Tools

The projects run under UNIX on Sun and RS6000 workstations. Tools are available to support most of the project work. Although the students may use their favoured editors, compilers, debuggers, and text systems, some tool usage is mandatory:

- E-mail is used for inter- and intra-team communication.
- WWW should be used for global announcements.
- S-CASE® has to be used for Object-Oriented Analysis and Design.

C++ is used as programming language, since the students already know C and/or C++ and S-CASE® supports the generation of code stubs in C++. A programming environment (with extensive browsing capabilities) is provided to support the edit-compile-debug cycle. CVS is recommended for version management. We furthermore provide a GUI toolkit to support user interface prototyping and a DPS to produce all the deliverables.

The students may use further tools if they want to. They may also use third party or embedded packages (e.g. class libraries), but only if acknowledged by the lecturer.

4 Some Notes on Good and Bad Projects

One thing that turned out to be critical for the success of the course is the selection of the right type of project. Up to now we tested the following kinds of projects: Editors, inventory systems, accounting systems, games, and simulations.

Inventory and accounting systems are proposed very often as examples for typical students projects. Our experiences are not very promising, especially when using object-oriented approaches. The basic architecture of inventory and accounting systems follows the same pattern. There is a simple database to keep track of some kinds of stocks or states of accounts. The other component is the user interface. Since there usually is no real processing logic, the user interface can be directly build upon the database. Since data modeling and (probably graphical) user interface design are important issues, there usually do exist special courses on databases and user interfaces. To experience behavioral aspects of software systems such projects are almost useless, since the data is passive.

Modeling object behavior is a very important aspect in all object-oriented approaches. There exist a wide range of approaches to model behavior which cannot be taught adequately in inventory and accounting systems. In such kinds of systems the application of use-case analysis, scenario modeling, state transition diagrams, etc. usually is not needed to understand the systems behavior. The students therefore avoid the usage of these techniques, since they seem to add complexity to the problem instead of reducing it.
The other types of projects offer much more opportunities for new experiences. They all have active processing components, like formatters in editors, the rule base in a typical game, and all the different kinds of objects in a simulation. Games and simulations have two additional advantages: They are more critical to changes in the requirements. It is therefore much easier for the students to experience the importance of design and the planning for change. Second and not less important, games and simulations are more fun to produce. Examples for actual project proposals are available on WWW (URL: http://www.cs.umu.se/tdb/kurser/TDBC10).

5 Experiences and Lessons Learned

Although the students already have some programming project experience, this is the first course where they have to work through a whole project in a team. That there are several teams pursuing the same project makes this kind of project even more complex.

The most critical deliverable in each project is the design document. It is the reference document for the subcontracting team, which has to produce a part of the prototype implementation. In this stage of the project the students experience the importance of earlier deliverables, since a lot of questions can be answered consulting these documents. A big problem for a subcontracting team is to withstand the impulse to change the design document according to their own ideas.

Changing the contract, i.e. the design document is no problem as long as both teams agree on the changes. This is especially true if these changes are going to affect class interfaces. Unfortunately, subcontractors tend to introduce changes without even notifying the other team. Due to limited inter-team communication most of these changes were detected too late. We are now going to tackle this problem by extending the scope of the formal meetings and through limiting the scope for subcontracting.

Initially the formal meetings were introduced to improve communication between the members of the same team. Now the teams shall use formal meetings also to coordinate changes with one another. In our first courses a team subcontracted a complete (detailed) design for prototype implementation. But a complete design may be too complex for the subcontracting team to understand, or it may simply be too “bad” to be implementable. In both cases the subcontracting team may be forced to completely redesign a system. This fall we changed the scope for the subcontractor. The subcontracting team now implements only a limited section (i.e. a subsystem) of the design. This has the effect that two teams have to work together for prototype implementation. This should make the students aware of the problems with changing contracts as described above.

Apart from choosing the right type of project, there originally was another problem related to project selection. Should all teams run the same project, or is it better to have
a different project for each team? Both alternatives have their pros and cons. With different projects there is no danger that teams work together and produce similar designs. Surprisingly the student teams proposed very different solutions in such situations up to now. With different projects the risk is minimized that teams try to implement their own system, using "we did a better design anyway" as an excuse. Having the same project for all teams obviously simplifies the correction of all the deliverables. Up to now both approaches worked fine. We therefore see no reason to restrict the project proposals in this way.

For games there usually exist some sort of description of the rules. An interesting lesson learned was that the teams performed differently on such projects with descriptions than on projects they proposed themselves. The emphasis in the requirements documents were on completely opposite topics. For games the teams produced extensive and detailed descriptions of the functional requirements and the user interface. For own proposals the emphasis often is on the non-functional requirements, because the students think that most of the other requirements are "crystal-clear." In this stage of the project it is very important to make clear that the teams do not write the requirements documents for their own pleasure, but that these are part of their contract with the customer, i.e. the lecturer.

Our experiences show that the students have a lot of difficulties with planning and communication. We therefore introduced a project schedule as an additional deliverable. The minutes of the formal meetings are now part of the final report. These additions have the effect that students now take planning and communication more seriously. It is our believe that planning and communication are impossible to teach in the classroom alone. Without project orientation there is no chance for the students to even recognize that these are probably the biggest problems.

6 Conclusion

Up to now we had 11 teams in three courses and still have difficulties to clearly define when a project is successful. Since the course lasts only 12 weeks the students must know that we do not expect a finished system after this time. The main goal of the course is to define a project which leads to a working prototype. Since the students propose systems themselves, it is also on them to define reasonable goals. It is the lecturers job to evaluate the deliverables carefully and adjust their scope to ensure feasibility.

Our current course is designed for third year computer science students. They already have some experiences with project work, where they have to implement a more or less clearly defined part of a bigger software system. In the software engineering
course the main message is that software engineering is much more than putting together some programmers to work on the same project.

Acknowledgements

Besides own experiences, this course builds on information about a similar course held on Worcester Polytechnic Institute, Worcester, MA, USA. I thank Lisa Taikitsadaporn for sharing her experiences on this course.

S-CASE® is a trademark of MultiQuest Corporation.

References


