Abstract

This paper describes our experience of teaching a Software Engineering course based on the use of group work to teaching and practise theoretical concepts. Our work has three main aims: firstly, for the students to develop a complete software project (from specification to delivery); secondly, to acquire knowledge of the main problems of group work and how to overcome them; and finally, to underline the importance of documenting software projects effectively. The following is the programme that we have developed to achieve these goals.

1. Introduction

Group work is vital for the development of large software engineering projects. Putting together a group that works effectively is therefore a critical management task. There are a number of factors that influence group work [1]: (a) group makeup; (b) group cohesion; (c) group communications; and (d) group organization. Good groups cannot guarantee the success of a project, but bad group work usually results in project failure. So we believe that group work is valuable experience for students, helping them to carry out their future professional work as Software Engineers successfully.

Group work also provides an excellent framework for teaching many Software Engineering topics, in that it allows most of the theoretical issues involved to be practised [2], [3], [4]. So, our efforts focused on teaching some theoretical issues using group work experience with students developing a Software Engineering project.

This paper deals with how we tried to achieve this. In our opinion this is a very difficult task for teachers, due to the fact that a complete large project cannot be developed in a semester, and the real problems associated with this development cannot be experienced. Moreover, we mean each group member to act, not as a specialist focused on a concrete task, (for example, programming, design, or tests), but working in several different tasks instead, and thus learning from all of them. In spite of this very difficult challenge, we developed a course in Software Engineering at the University of Alicante that seems to be extremely satisfactory for both students and teachers.

Large projects are often ‘one-off’ and suffer from unique problems, but they have in common the need for group work and for effectively documenting the software process. So the course that we developed is based in group work, and has three main objectives:

- To develop a complete project by teamwork
- To show that effective group work is a very difficult task, and learn from this experience, and
- To show the importance of system documentation in the development of a large software project.

We have structured the rest of this paper as follows. Firstly, we present the Software Engineering related courses at the University of Alicante (in Spain), in order to justify the contents of the Software Engineering course that we have developed.
Secondly, we analyse initial questions related to our decisions about the size of the work groups, the methods and tools that we have used, and we give an example of a project proposal that must be developed by the students.

We then explain the life cycle model that we used, and the planning of the tasks that must be carried out, the programme schedule, and the deliverables associated with each milestone defined in the plan.

We show the results obtained from our experience of applying our custom life cycle model, and we finish by explaining some conclusions we have reached from our work.

2. Software engineering at the university of Alicante (in Spain)

At the University of Alicante (http://www.ua.es), students can obtain a five-year graduate degree in computer science called "Computer Engineering". The body of Software Engineering knowledge is mainly obtained from three 60-hour courses. In the first, students learn about requirement specifications, and different methods used to analyse software engineering projects. The second is mainly focused on the study of software project design, and the application of several design methods. The third and last course is the one dealt with in this paper. Instead of analysing a single topic in depth, maintenance or validation for example, we intend to complete students’ knowledge of Software Engineering. So we refer to several topics that we integrate with the knowledge that students have already obtained from the first two courses.

We divide the course in 30 hours of theory, and 30 hours of practice. In the theory classes, we summarize the main concepts of the following topics: Planning activities, the testing process, configuration of the management process and the maintenance process.

The course website is at: http://dccia.ua.es/dccia/inf/asignaturas/IS2. The site contains the slides used in the theory room, and other information, such as the bibliography, related papers and links.

The course is taught over four hours per week: two theory hours, and two practical hours, spread over 15 weeks. We explain the concepts in the theory classes and we practice them in the practical classes (except for some of those related to the maintenance process). For each concept, we broadly explain their importance, the methods and tools used to apply it, and how it should be applied depending on several factors, such as the type or the size of project, the resources available, etc. We think that: "A learned concept is much more valuable if we know how to apply it to a real problem". This therefore, is the premise that guides our teaching of software engineering.

3. Preliminary factors

Before we explain our laboratory programme, we would like to present some preliminary factors such as: the size of work groups, the methods and tools used, and the project proposal.

3.1. Group size

We have opted for five member work groups. This number was chosen in order to facilitate organisation of the groups. This organisation is a result of our aiming to involve each member in all tasks, thus learning from all of them. The role of each group member is defined from the beginning, and is explained to the students in the first practical class. Basically, each member plays a twofold role: (1) he/she acts as the configuration manager of a document that must be delivered at the end of course; and (2) he/she acts as a subordinate of the other configuration managers and contributes to create the document.
that must be delivered. The roles that we have defined are contributes to create the corresponding document that must be delivered. The roles that we have defined are:

- Planning configuration manager (P),
- Design configuration manager (D),
- Source code configuration manager (C),
- Test configuration manager (T), and
- User Manual configuration manager (U)

Each is responsible for developing the corresponding document: planning, design, implementation, test, and user manual. We have created documentation standards in order to control document structure and presentation. We provide templates for all documents, and set the precise contents of each one. Each configuration manager is responsible for the following tasks: (a) allocating and distributing the contents of the documents to the other four group members; (b) integrating the work of group members into the whole document and delivering it to teacher; (c) controlling document changes and quality; (d) writing a final report whose contents depend on the configuration manager type. The final reports contents are as follows (P): makes a real schedule of the project and states the differences with the initial draft schedule; (D): states the match between the design and the source code delivered; (C): reports the correlation between logical and physical design; (T): reports the results of unit testing and the validation testing process; (U): states the match between the requirements and the design. These reports are delivered at the end of the course by each configuration manager (all students).

Besides those described above, each configuration manager must carry out the following tasks: (P) monitor and control the planning project during the development process; (D): monitor and control all design changes during the development process; (C): develop the corresponding implementation standards, and design the integration process; (T): design the drivers and stubs used in tests; (U): monitor and control any changes to the initial specifications during the whole development project process.

The teacher does not set group communications, and each group can decide how to exchange information on the status of their work, the design decisions that have been made, changes to previous decisions and any other information. The group communications implemented are reported in the Planning document.

3.2. Methods and tools

We use an object-oriented approach to develop the proposed project. We have chosen UML (Unified Modelling Language) [5], in order to specify, visualize, construct and document the software artefacts of the project. The reason for this is that this language is widely used by object-oriented developers [6]. The extensive use of UML has led to the creation of CASE tools that use it and generate source code automatically from design. An example of this kind of tool is ‘Together’, which we have used in order to make students work easier.

It is important to mention that we assume that the students have systems analysis and design knowledge and are therefore familiar with object-oriented design using UML from previous courses. Basically, we use the following diagrams in the Design document: use case, sequence and class diagrams. The use case diagrams provide a means of describing the external view of the system and its interactions with the outside world. The sequence diagram model shows the dynamic aspect of the system while emphasizing the ordering of messages in time. They are used to illustrate the scenarios associated to each case used. The class diagrams show the static structure of the system, the internal structure, and also the relationships between classes and inheritance structure.
We also use Together Control Center v. 6.0. This is a modelling tool developed by TogetherSoft Corporation (http://www.togethersoft.com), which fully synchronises modelling diagrams and source code. This tool also includes all features needed to build enterprise level applications, allowing the entire development team to collaborate using a common language, diagrams and software. We can model the system using UML and generate code automatically in Java. It also provides very useful features such as: a user interface builder, a version control of artefacts created, and a testing framework. The User Interface Builder allows a graphical user interface (GUI) to be designed and built for a Java application. The testing framework allows the development of both unit tests for source code and visual tests for user interface components.

3.3. The project proposal

We now describe an example of a project that we propose to the students. This project has to be developed by each group of students. The project consists of developing an automatic tool to create a calendar from input data related to the type of process model used, and the project activities.

The input data consists of the process model used (waterfall model, incremental model, spiral model), project activities, including their duration, precedence, type of resource and number of units of this resource that each activity requires. It also includes the data from which to construct the calendar and the cost of use of each resource by unit of time.

The output data consists of two possible Gant charts: an activity bar chart and a staff allocation bar chart. For each activity they show the earliest and latest start times, and the earliest and latest finish time. The financial cost of the project is also stated.

We identify the following functionalities: (F1): Choose the process model, this will show a template with the process model chosen; (F2): Introduce the input data, the user can eliminate process model activities or subdivide them into other sub-activities; (F3): Generate the calendar and calculate the financial cost; (F4) Manage data calendars: the user can store the data calendar, restore a previously saved data calendar from another project, or modify it; (F5) Resource manager: the user can add, eliminate or modify resource types used by the project.

This proposal is given in more detail to the students. We allow each work group to make minor changes to these initial specifications.

4. The programme

We have defined a specific process model in order to achieve our objectives of using group work to learn software engineering concepts effectively in the practical hours.

The proposed model has three sequential phases that we have called: user interface centred, basic functionality centred and advanced functionality centred. We will now describe each of them in detail.

4.1. Phase I: user interface centred

The objective of this phase is to obtain a non-functional prototype so as to validate the specifications with the end users. Additionally, as a result of this phase, the students must understand all parts of the system directly related to the user.

Each work group must draw up a Planning document that includes a system specification, the estimated project costs, a project risk analysis and a draft project calendar.
They also draw up a *Design document* centred only in the *user interface* design aspects. This document includes cases and scenarios that will be used to validate the specifications and also to validate the finished product. This document also describes the class diagrams related to user interface design.

Finally, the work group must implement the non-functional prototype, integrating all components designed during this phase and correctly documenting the source code obtained in the corresponding *Implementation document*. This phase lasts 12 hours (6 weeks).

### 4.2. Phase II: basic functionality centered

The input data is validated during this phase. The group proceed with the design of the data structures that are needed to implement the project. They must also implement the following "basic functionalities": F1, F2, F4, and F5. So, at the end of this phase, the user can input data, which is stored in data structures that will be used to create the calendar. During this phase, the students also acquire more knowledge about the project before implementing the functions that produce the result required by the user.

Each work group must draw up a *Design document* that reflects the program architecture used to support the basic functionalities that are to be implemented. This document contains the new classes created and their relations with: (a) the classes designed in the previous phase referring to the system interface, and (b) the internal data structures created that are used as the basis for implementing the functionalities of the following phase.

They also elaborate a black box design of basic functionalities F4 and F5, which will be implemented at the end of this phase, recording possible incidences when carrying out test cases. This black box design and the results obtained are reported in a *Test document* that has to be delivered to the teacher by the corresponding configuration manager.

Finally, the work group has to implement the basic functionalities and integrate them with the prototype obtained in the previous phase. Additionally, the source code must be adequately documented in an *Implementation document* and delivered. This phase lasts 10 hours (5 weeks).

### 4.3. Phase III: advanced functionality centered

This is the last phase, in which the project must be completed. The group designs the algorithm that implements the functionality F3, which calculates the calendar derived from the input data, and its associated cost.

Each work group must elaborate a *Design document* that includes the possible new classes defined and their relationships with the classes defined in previous phases.

Once F3 is implemented, it must be integrated with the results of phases 1 and 2, and reported in an *Implementation document*.

The functionality F3 must also be white box tested. The test cases obtained are implemented and any possible errors found are debugged. All this information is put into a *Test Document*.

The work group has to create a *User manual document* that reflects the correct use of the program by the possible users. Finally, we can validate the program with the teacher. This phase lasts 8 hours (4 weeks).
5. Evaluation of the proposal

Implementation of the above-mentioned proposal planning for phases I, II and III demands a great effort by the teachers, in that they act as high level managers of the project for each work group. So, they must monitor and control the work carried out by the students at all times.

We have defined three key milestones for the project, associated with the end of each phase. At each milestone, the configuration managers must deliver the corresponding documentation. The teacher then evaluates the work carried out by the group.

The score obtained for each phase depends not only on the quality and contents of documents, but also on other factors such as the attitude of group members, and their ability to overcome problems caused by different personalities in the group.

The teacher scores each phase from 0 to 10 points for each group. Each phase has been weighted differently in order to obtain the final score for the practical hours. Therefore, we have assigned weights of 15%, 20% and 35% for phases I, II and III respectively. The students also have to answer some questions about the development of the project by their work group individually. These questions are weighted as 30% of the final score. So, the final score for the practical hours is:

\[(0.15 \cdot PI) + (0.2 \cdot PII) + (0.35 \cdot PIII) + (0.3 \cdot \text{questions}).\]

The course score is obtained by calculating the average of the score corresponding to practical hours and the score from the theory hours (a written test). A minimum of 5 points must be obtained in both of them.

6. Analysis of the model proposed

The model proposed for the practical hours in the section 3, as we have previously stated, implies a great effort for the teacher involved, if he has to evaluate several groups. For example, we have experimented with practical sessions (2 hours) with 20 students, meaning four work groups. He has to monitor the work of students continuously and he must mediate if any conflict occurs in a work group. When a milestone is reached, he must evaluate the documentation delivered in a very short time and report the results and the group score to each configuration manager. This means that he has to talk to each configuration manager and explain the errors found and how to solve them, or even recommend that some parts of the work be done again. If the teacher has monitored the work of students correctly before delivery, there is no reason for this evaluation task to be very "hard". This effort will be compensated by the advantages that we feel the model proposed offers.

The evaluation carried out by the teacher allows students to learn actively, in that there is feedback on the work developed by the work group. As each milestone is reached, the students are informed of which parts of their work have to be improved in the next phase, or, if the worst comes to the worst, to be repeated. This allows the student to benefit from this feedback and learn effectively.

Each group suffers from the problems associated with group work, such as poor communication, bad coordination or poor work carried out. They also suffer from a serious problem relating to the different personalities in the group. The teacher leaves it to the students to decide which group they want to belong to. There are groups in which the personalities are complementary, but there are others in which the personalities are totally incompatible. As a result of group makeup, a particular group may not be able to overcome the difficulties that crop up and the group cannot develop the complete project, or some members of the group may decide to leave it, etc. These difficulties have to be
borne in mind by the teacher when evaluating them. Although the project cannot be completed due to the factors outlined, we consider that this is a very positive factor for the students, who can learn in spite of having carried out an unsuccessful project.

The students learn gradually, in each phase they focus on one part of the problem and are better able to solve it. Phase I is focused in "what" the problem is, and the students’ efforts are focused on understanding the requirements of the end user.

We have noticed that this first phase has a lesser weight associated because the students are beginning to integrate themselves with the group and understand "what" has to be done. The tasks accomplished in each phase are very similar, although they focus on different issues. So, in phase II the students become more familiar with the process model and they can begin to think about "how" to solve the problem. This phase has a medium weighting. Finally, the third phase has the greatest weighting, in that the students have had overcome two previous milestones and have received the necessary explanations from the teacher about the work carried out. This phase is exclusively focused in "how" obtain the outputs required by the user, and the student are less worried about the process model.

Finally, the students verify the importance of documenting the software project effectively. As a consequence of group work, and the process model proposed, the students must consult documents written in previous phases or written by others members of group. If the document looked up is, for example, bad quality, too extensive or the content is less valuable, this represents a problem that affects the work that must be carried out. As this situation occurs for all group members, they try to create useful and good documents in order to increment their productivity as a result.

7. Analysis of the results obtained

We have applied the model proposed over the last two courses. We have tried several formulas for teaching software engineering with work groups since 1995. We reported three different experiences in 1998 at a national congress [7], and here we propose a new model proposed that we have applied over the last the two courses (2000/01, and 2001/02). We have approximately 140 students per course. This means seven weekly sessions with 20 students each (four work groups). Approximately 98% of the students finish the course successfully. External factors oblige 2% to abandon the course.

Of this 98%, in general the students obtain a high degree of satisfaction due to the fact that they have learned in spite of any problems that may have arisen. They appreciate their experience in that they practice the theoretical issues that they have learned in the theory hours, and other associated software engineering courses. 22 (out of 28) work groups finished the proposed project completely during the 2000/01 course, and 26 during 2001/2002. This is due to the teachers having more experience, in that they also learn from difficulties encountered by students in previous courses and can guide the work of students more effectively. The students were asked to state what were the main difficulties encountered. 90% of them stated that the main problem, which they faced, was coordinating group work appropriately; this was mainly due to the presence of different personalities in the group. 7% of the students considered that the main problem was their lack of experience with the tools used, something that reduces their productivity, and 3% of the students felt the main problem to be the tight schedule.

With respect to the time spent by the students carrying out the proposed project, 65% of the groups needed to work at home two extra hours per week in order to complete the project, 20% of the groups needed to work one extra hour, and 15% stated that they did not need to hold meetings outside the practical hours. 96% of the students appreciated the process model used, in that it clearly defined the tasks that they had to perform, and 93%
considered the documentation standards provided by the teachers to have a positive influence on group productivity.

99% of the students considered that group work turns out to be more difficult than they had imagined initially in the theory hours, and so they consider the practical hours to be a necessary experience. Another interesting fact is that the students appreciated their experience, not only immediately, but also much more so when they started working as software engineers. Students often say to the teacher: "in my job, I have the same problems that I encountered (to a lesser extent) in the practical hours of the software engineering course". So the teachers also experience a high degree of satisfaction.

Finally, we underline the fact that this high percentage of successful groups is affected by a fact that the teachers have noticed: the students learn that the success or failure of the group depends on the work of all members and on the degree of cohesiveness achieved, favouring "ego-less" programming [8]. Each member can influence the score achieved by the group, and this fact encourages the group members to cooperate actively in order to contribute to the success of the group instead of its failure.

8. Concluding remarks

We have explained our experience of teaching software engineering by using work groups. We have proposed a model for teaching software engineering that includes: (a) a specific group organization, and (b) a specific process model that allows us to achieve three main goals: (1) for each group to develop a complete project, (2) to experience the main problems associated with group work and to learn from them, and (3) to show the importance of documenting software projects effectively.

The model proposed has been applied at the University of Alicante over the past two years. Our experience reveals a high degree of satisfaction among both students and teachers. Among the positive aspects of the model proposed we highlight the fact that it favours: (a) active learning; (b) incremental learning; (c) learning in spite of failure; (d) learning from experience.

10. References