Teaching Software Engineering using Globally Distributed Projects: the DOSE course

Martin Nordio
ETH Zurich, Switzerland
martin.nordio@inf.ethz.ch

Carlo Ghezzi
Politecnico di Milano, Italy
carlo.ghezzi@polimi.it

Bertrand Meyer
ETH Zurich, Switzerland
bertrand.meyer@inf.ethz.ch

Elisabetta Di Nitto
Politecnico di Milano, Italy
dinitto@elet.polimi.it

Giordano Tamburrelli
Politecnico di Milano, Italy
tamburrelli@elet.polimi.it

Julian Tschannen
ETH Zurich, Switzerland
julian.tschannen@inf.ethz.ch

Nazareno Aguirre
University of Rio Cuarto, Argentina
naguirre@dc.exa.unrc.edu.ar

Vidya Kulkarni
University of Delhi, India
vkulkarni@cs.du.ac.in

ABSTRACT

Distributed software development poses new software engineering challenges. To prepare student for these new challenges, we have been teaching software engineering using globally distributed projects. The projects were developed in collaboration with eleven universities in ten different countries in Europe, Asia, and South America. This paper reports the experience teaching the course, describing the settings, problems faced organizing the projects and the lessons learned.

Categories and Subject Descriptors

D.2.9 [**Software Engineering**]: Management— *Programming teams*; K.3.2 [**Computing Milieux**]: Computers and Education— *Computer and Information Science Education*

General Terms

Management, Documentation, Human Factors

Keywords

Distributed software engineering, multinational project

1. INTRODUCTION

Today's software production is increasingly distributed. Gone are the days of one-company, one-site projects; most industry developments involve teams split over locations, countries, and cultures. This geographic distribution poses new challenges in communication, software requirement specifications, interface specifications (in the sense of program interfaces, also known as APIs), and project management.

ICSE '11, May 21–28, 2011, Waikiki, Honolulu, HI, USA

One of the roles of software engineering courses is to prepare the students for software engineering development as is done in industry. It is indeed a widely adopted practice to include a development project as part of software engineering courses, so that students get exposed to the problems of the field, and get practical experience on some approaches to deal with them. These projects are typically used to teach concepts such as requirements, interface specifications, design, etc., via a "hands-on" approach.

We have experimented with teaching software engineering using a geographically distributed software project involving various countries with different cultures, native languages and time zones. Such a project gives the students the opportunity of facing the challenges of distributed software development; it also helps them understand typical software engineering issues, such as the importance of software requirements specifications, or the relevance of adequate system design. This project provides an authentic distributed development experience, in which many of the difficulties typical of such a setting take place.

Despite the mentioned benefits of running a distributed project, there exist various difficulties that make the organization of the project itself as well as the corresponding courses (in their respective countries) challenging. Besides the obvious time scheduling inconveniences, caused by courses being run in different countries with different time zones, a particular problem to tackle is the *coupling* of the participating teams. The success of the project does not only depend on the success of a (local) team, but also depends on its partner teams, located in different countries, being successful too. Issues such as how to keep teams committed to their peers, and how to handle risks and failures are part of the experiences that we report in this paper.

In this paper, we describe our experience teaching distributed software engineering centered in the above described distributed projects, and involving, in the past four years, eleven universities in ten different countries in eight different time zones. We explain how this experience enabled us to recreate the atmosphere of an internationally developed project, the obstacles we faced, and how we dealt with these.

This paper expands the experiences reported in [8] with new experience of distributed software development gath-

ered in the courses taught in 2009 and 2010, which involved more universities and a more ambitious project.

2. THE DOSE COURSE

The Chair of Software Engineering at ETH Zurich has been teaching a "Distributed and Outsourced Software Engineering" (DOSE) course for several years. The course targets master students with good experience in programming and some prior knowledge in software engineering. The topics covered in the course are requirements elicitation and writing software requirements specifications, the CMMI model, quality assurance, cost models for outsourcing, supplier agreements, and risk management in distributed projects.

To face the challenges of distributed software development, the course has incorporated, in the last four years, a distributed project. To make the course manageable, baring in mind that these distributed projects pose new challenges both for the students and for teachers, the topics of the projects are carefully selected to be simple, yet challenging enough to keep students motivated. In 2010, the project was developed in collaboration with eleven universities: (1) ETH Zurich, Switzerland; (2) Hanoi University of Science and Technology, Vietnam; (3) Korea Advanced Institute of Science and Technology, Republic of Korea; (4) Odessa Polytechnic National University, Ukraine; (5) Politecnico di Milano, Italy; (6) State University of Nizhny Novgorod, Russia; (7) University of Debrecen, Hungary; (8) University of Delhi, India; (9) University of Rio Cuarto, Argentina; (10) University of Zurich; and (11) Wuhan University, China.

Each participating university has its own course with its own course material, however, the covered topics are similar. The course material used at ETH Zurich is shared, and the lectures at ETH Zurich were recorded and were available for all students [1]. Eiffel and Design by Contract are used as a general framework for the analysis, design and implementation of the projects.

2.1 Setting up a Distributed Project

Setting up a distributed project for the first time is not an easy job. Many unpredicted problems could rise in the last minute: misunderstandings in communication, delays in finishing the assignments, integration problems, etc. An important issue to consider is how to keep the students' motivation high, and how to avoid students leaving the course.

The first time the DOSE course integrated a distributed project was in Fall 2007, in collaboration with University of Zurich, ETH Zurich, Odessa National Polytechnic University, and the State University of Nizhny Novgorod.

The project topic was the development of a system to analyze e-mail postings of computer science events, from mailing lists such as the ECOOP list and SE World, to feed the Computer Science Event List (CSEL). The automatic part of the system identified key elements of a conference announcement, such as event name, event date and call for papers deadline, to prepare a CSEL entry. Since the identification cannot be perfect, the system includes a human editing step to correct mistakes in the automated identification process. The system was divided into three subsystems: (A) ANALYZE: automatically extract the essential information; (B) BEFIT: user interface for interactive correction; (C) COMBINE: integration of components A, B and the CSEL website.

To reduce the communication overhead, which we assumed would be a major source of potential obstacles, in this first experience an instance of the system was implemented by three teams located in two countries. Typical group configurations were (A) Zurich – (B) Nizhny Novgorod – (C) Zurich; and (A) Zurich – (B) Zurich – (C) Odessa.

This approach has a pedagogical benefit: it forces the teams, in their work and especially their interactions with other teams, to focus on APIs. This is a key software engineering concept; the best way to teach it is by experience, as students discover the essential role of high-quality interface descriptions, in particular contracts, and realize the extreme degree of precision required to avoid mishaps. The pedagogical value is higher and the lessons deeper than what can be learned from the experience of implementing someone else's blueprint in the process-based approach (although that experience is also useful).

The project, developed over 11 weeks out of the semester's 13, was divided into four phases:

- Phase 1: Requirements (4 weeks).
- Phase 2: Interface specification (3 weeks).
- Phase 3: Implementation (2 weeks).
- Phase 4: Testing (2 weeks).

While the project was relatively simple and small, the resulting system implementation was weak. The main problem in the project was the delay in finishing each phase. These delays were produced by various issues, such as misunderstandings in the APIs. These resulted in the impossibility of integrating the final system in some of the projects. According to the students' feedback, one of the main lessons they learned from DOSE 2007 was the importance of APIs.

In Fall 2008, Politecnico di Milano and University of Debrecen joined the course. The main difference with respect to 2007 was that each project was globally distributed in three locations; typical group configurations were: Zurich – Nizhny Novgorod – Milano; and Debrecen – Milano – Zurich; and Milano – Zurich – Odessa

The project was organized in phases similar to those of 2007, but 13 weeks of the semester were dedicated to the project. To avoid misunderstandings and ambiguities of the requirements document, the interface specifications were written in Eiffel using contracts (our previous work [9] shows a study of the use of contracts in distributed projects). The last phase was the implementation and testing of the system.

In DOSE 2007, we observed that students typically started working on the assignments at least a week after these were handed out. This showed that students were not fully aware of the criticality of deadlines in distributed software projects, and produced integration problems (the assignments were finished too close to the deadline, leaving no time to sort out integration issues). To solve this problem, in 2008 each phase was implemented in two cycles; the students had to submit a draft version halfway through an assignment. The introduction of two deadlines per assignment resulted to be a key element that contributed to the success of the DOSE projects.

2.2 A Challenging Project

The next two editions of DOSE included universities in Asia and South America, increasing the challenge of people of different cultures and time zones working together.

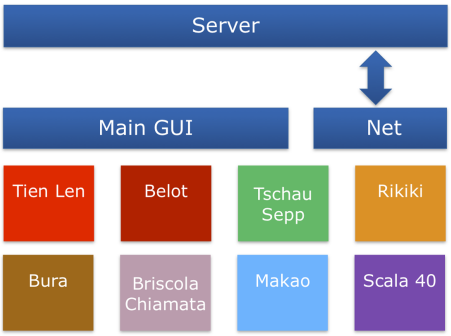

Figure 1: Architecture of the DOSE 2009 project.

A main difference with projects from previous years was that, in order to make the setting more realistic and challenging, as well as more motivating for the students, we chose a larger project, to which all students contributed, as opposed to developing various independent instances of a smaller project, and in previous editions of the course. Still, in order to make the system realizable and keep the risks of the whole development to a minimum, the design of the project to be developed needed to be easy to partition, into relatively independent subsystems.

For example, in 2009, the overall project consisted of a game platform where players could log in, and choose a game to play. Figure 1 shows the architecture of the project.

The students developed eight networked multi-player card games, with each game being implemented by two teams located in two different countries. For each game, one team implemented the logic of the game and the other team implemented the graphical user interface (GUI) and the network communication. The implemented games were *Tien Len* (Vietnamese game), *Belot* (Bulgarian game), *Tschau Sepp* (Swiss game), *Rikiki* (Hungarian game), *Bura* (Russian game), *Briscola Chiamata* (Italian game), *Makao* (Polish game), and *Scala 40* (Italian game). The games Belot and Makao were proposed by students originally from Bulgaria and Poland.

The main changes in the structure of the project, compared to 2007 and 2008, were the introduction of a scope document and a communication plan. In the first week of the course, we provided a scope document describing the general architecture of the game platform. In phase 1, the students developed one scope document per game describing the scope of each project, and the role of each student in the team. The scope document helped the students to avoid misunderstandings regarding the scope of each component. The project was organized as follows:

- Phase 1: Scope document and communication plan (2 weeks)
- Phase 2: Requirements document (3 weeks)
- Phase 3: Interface specification (2 weeks)
- Phase 4: Implementation and Testing (6 weeks)

The project for the 2010 edition of DOSE consisted in the development of a learning platform to help users to learn languages such us English, Spanish, German, etc. The project was again organized in a single system, developed by 110 students. The system was composed of eleven subcomponents (each subcomponent developed by one group). A group consisted of three teams located in three different countries. The main reason to split the project in three different locations was to experience more about the barriers of developing software using different time zones.

3. ASSESSMENT

The project outcome in 2007 was not good: several groups had problems to integrate their system and in the end only one team partially integrated the system. In DOSE 2008, 2009, and 2010 the situation improved.

In 2008, the results of the project were satisfactory: most teams integrated the respective subsystems and managed to have their projects working by the end of the semester. Although some functionalities of the project were missing, the requirements with the highest priority were implemented. At the beginning of the semester, there were six projects (42 students distributed in the five universities): five projects were implemented and successfully integrated; and one project failed. On average, each of the implemented instances of the project had 44 classes, and 4800 lines of code (the projects were developed by 6-7 students).

In 2009, the results were even better. All the projects were successfully implemented and integrated in the game platform. The whole project has 300 classes and 55'000 lines of code; each group implementation has an average of about 37 classes and 6800 lines of code. Each group had 7-8 students.

We achieved a similar level of success in the 2010 edition of DOSE, despite the increase in the people involved in the development. The eleven groups successfully implemented the project. The whole project consisted of over 600 classes, and more than 130'000 lines of code (on average, each group implementation has 55 classes, and 12'000 lines of code). Each group had 7-9 students.

4. ORGANIZING DISTRIBUTED PROJECTS

4.1 The Challenges of Distributed Projects

4.1.1 Interface Specification

Interface specifications are an important concept in software engineering. This concept is taught in any software engineering course. Distributed projects make this concept even more relevant: a bad design in the API might result in the failure of a project.

In DOSE 2007, many issues arose due to the lack of an appropriate design of the API. For example, one group implemented a class EVENT to represent conferences, workshops, symposiums, and summer schools. This class stored the name of the conference, url, abstract and submission deadlines, and the date and place of the event. This class had two different semantics for the teams, which were located in different countries. For team A, this class modeled an event where all the data is valid. For example, a restriction is that the date of the abstract deadline should be earlier than the date of the submission deadline. Team B assumed that the class EVENT represents any event even if the data is not valid. According to the latter interpretation, one would be able to create a conference where the dates are

not valid. This assumption and misunderstanding between the teams introduced problems during the implementation phase. These problems introduced delays in the development of the projects, which in some cases led to not finishing the project.

A way of dealing with API specification problems would be to take the idea of providing a partitioning of each subsystem into modules (e.g., the above mentioned partition of each game subsystem into logic and GUI) further, and also force students to attain to a given API for each subsystem module. But as we mentioned, interface design is a crucial software engineering concept, that we want students to get training in. So, we want teams to design their own APIs, but help in avoiding API misinterpretation. We provide then a relatively simple API design process, based on code reviews, and more importantly, design by contract. The students first write the API for the core parts of their systems. This API has to be fully equipped with preconditions and postconditions. This forces the students to clearly state the assumptions of each class and method, which is an important part of the API design. Then, the students do a mandatory code review for the API design. The members of the team have to review the API and integrate the feedback before starting the implementation. Furthermore, the teams have to define a procedure for changes in the API. Thus, changes will be communicated to the whole team describing how to adapt the code to the new API.

The focus on a better API design are one aspect that helped produce better results in the projects from 2008-2010. Also, our case study in distributed projects [9] has shown that contracts helped to develop interface specification and to clarify misunderstandings in many ways. For instance, the usual misunderstandings of textual specifications due to natural language ambiguity are increased in the context of our distributed projects, since students come from different cultural backgrounds, and many are not fluent in English (the language used for interaction). Contracts helped in alleviating this issue.

4.1.2 Project Management

In the DOSE course, students are exposed to the problem of project management. Each group has to manage and organize its project monitoring the progress and communication between the teams. In the first edition of the course, we did not impose any scheme for the organization of the project: each group had the freedom to organize the project as they preferred. The lack of a person playing the role of project manager in the group made the organization of the project more difficult.

Since 2008, we impose the project organization for all the groups introducing a group leader (project manager) per group and a project leader per university. The role of the project leader is to discuss the project with the students and to help with the management of the project. Each project leader gives a weekly report of the progress of each team to the project management. This report helped to monitor the progress of the project, and also to detect problems. The role of the group leader is to organize his own group and report progress to the project leader in the corresponding university.

To further improve the organization of the projects, we provide templates and examples from previous years for the documents that are produced.

4.1.3 Communication

Communication is key for the success of geographically distributed projects. Our experience in developing distributed projects has shown that a communication plan for each project is a necessary condition for a successful project. At the beginning of the project, the students are requested to write a communication plan defining the contact persons in each team, a weekly time slot of one hour for meetings, and tools to use. This plan also describes a communication protocol indicating the expected time to reply e-mails, and how to proceed if there is no reply. Generally, the communication plans led to almost no communication issues being reported to the project leaders in 2009-2010, as these were solved within groups.

4.2 Student Fluctuations

Students might leave a course during the semester. When they are part of a distributed project, this not only affects their local team, but might have an impact on the teams of other universities. In 2007 and 2008, we experienced difficulties with whole teams leaving the project. The main reasons were lack of motivation or inability to cope with the language barrier.

To ease this problem, we require an initial team size of three or more students per team. In the case that during the course a student quits, the remaining team members can continue, possibly reducing the scope of their part of the project. In the case that a whole team leaves the course, the groups have to be reorganized or a subsystem of the project has to be dropped.

At Politecnico di Milano, the DOSE project is a selective part of a large software engineering course (with more than 100 students). Students have the option to implement a distributed project or a local project. Many students in Milano were interested in the distributed project, and the best applications were selected to participate. This selection produced an excellent result from the students at Politecnico di Milano: no student left the project, and the quality of requirements documents and the implementation was very good. Last year, we have applied a similar selection in Russia, Hungary, and Vietnam. This selection has resulted in an improvement in the quality of the result.

4.3 Trainings and Exercises

The DOSE project involves different universities with different curricula. To normalize the level of the students, we run trainings before the course starts. For example, in Russia and Argentina, we teach a short course on Eiffel.

During the course, we organize exercise sessions where the students face communication as well as project management problems. The goal of these exercise sessions is to emphasize the importance of good communication and project management. These sessions help the students identify potential problems in the group. Although these sessions are not mandatory for the students, they are encouraged to participate by the corresponding local teaching staff.

An example of these exercises is the on-line group competition that we organized in 2010. Each group consisted of eight students located in three different countries. Each member in the team had a role: A, B, C1, C2, C3, D1, D2, or D3; before the competition started, students had to select their roles in the group. During the competition, they were allowed to use any tool (Skype, e-mail, etc.).

When the competition started, each participating student received the role card by e-mail. Each card has an array of integers and a description of tasks to solve. The tasks were simple (for example, find the maximum element in the arrays C1+C2+C3) but they involved the array of the other team members. There were 17 tasks distributed in the eight role cards, some of them where repeated in several role cards. All role cards mentioned that the solution had to be sent by the person with role A. Only role card A mentioned that only seven out of the 17 tasks had to be solved. The role cards were designed in a way that if they distributed the tasks, and organized well, they would finish earlier. The role cards C1, C2, C3 contained the same task; to solve them, the students only needed the arrays C1, C2, and C3. The role cards D1, D2, and D3 were designed in the same way.

The first team completed the tasks in 41 minutes, and the last team in 71 minutes. While the tasks were simple, some of the groups reported wrong results.

The students' feedback reported that in some groups, the person in role A did not tell the others what tasks to solve; thus they solved all the tasks. Looking to the Skype logs, we found that chat messages were sent every 20 seconds. The students reported that at the beginning of the competition, it was difficult to work because of all the messages they got. Some of the groups created sub-groups with different chats. This organization reduced the overhead in communication.

The feedback also reported that the exercise was very interesting and they learned the importance of a project manager and good communication. We found that the exercise was interesting, and we plan to repeat it in the next edition of our course.

5. RELATED WORK

Lessons learned on educational experiences similar to ours have been reported [5, 4, 3, 2, 6, 10]. Gotel et al. [5] describe the lessons learned from the development of a project across three globally distributed educational institutions in the US, India, and Cambodia. A similar experience is described by Damian et al. [4]. They report on the teaching experience developing software requirements specifications in geographically distributed software development with three universities (located in Canada, Australia, and Italy), focusing on the times zones and the cultural differences. Bosnic et al. [2] present a teaching experience for distributed software engineering in collaboration with Croatia and Sweden; Bruegge et al. [3] develop a similar experience in collaboration with universities in the US and Germany. Our experience, in particular DOSE 2010, involved more universities and different settings; some of the groups had a twelve-hour time difference, others had a seven-hour time difference, and others only two-hour time difference.

Meyer et al. [7] have described our first experience in distributed software development, DOSE 2007. This paper extends our previous paper [8].

6. CONCLUSIONS

We have presented our experiences regarding an approach to teaching distributed software engineering centered in a distributed software development project. The approach has been implemented every year since 2007, and we have made various significant improvements based on the lessons learned in the earlier editions of the course. We described some of these improvements, with their corresponding motivations and results. Among these, the emphasis on API design, and the role that contracts play in this activity, is a key characteristic of the approach. We have also emphasized the development of communication skills, since we identified that at least 30% of the time spent by students in the project corresponds to communication.

Organizers in each university should ensure that students commit to the project. In many universities students can leave the course at any time; this is strongly undesirable for a distributed project. We have addressed this problem working with small groups of volunteer students. In some universities, students are selected. In the last year, we had only few students leaving the course. This allowed us to successfully reorganize groups when students dropped the course in the middle of the execution of the project.

7. ACKNOWLEDGMENTS

We would like to thank all the people involved in DOSE: Do Le Minh, Franco Brusatti, Huynh Quyet Thang, Lajos Kollar, Mei Tang, Natalia Komlevaja, Peter Kolb, Raffaela Mirandola, Sergey Karpenko, Sungwon Kang, Victor Krisilov, Viktor Gergel; and the students who took the course.

8. REFERENCES

[1] Dose 2010 course material and videos. `http://se.inf.ethz.ch/teaching/2010-H/dose-0273/index.html#slides`.

[2] I. Bosnic, I. Cavrak, M. Zagar, R. Land, and I. Crnkovic. Customers' Role in Teaching Distributed Software Development. In *CSEE&T*, 2010.

[3] B. Bruegge, A. Dutoit, R. Kobylinski, and G. Teubner. Transatlantic project courses in a university environment. In *7th Asia-Pacific Software Engineering Conference*, 2000.

[4] D. Damian, F. Lanubile, and T. Mallardo. Investigating IBIS in a Distributed Educational Environment: the Design of a Case Study. In *Workshop on Distributed Software Engineering*, volume 1, 2005.

[5] O. Gotel, V. Kulkarni, L. Neak, C. Scharff, and S. Seng. Introducing Global Supply Chains into Software Engineering Education. In *SEAFOOD*, 2007.

[6] M. Hawthorne and D. Perry. Software engineering education in the era of outsourcing, distributed development, and open source software: challenges and opportunities. In *ICSE*, 2005.

[7] B. Meyer and M. Piccioni. The allure and risks of a deployable software engineering project. In *Proceedings of the 21st IEEE-CS Conference on Software Engineering Education and Training*, 2008.

[8] M. Nordio, R. Mitin, and B. Meyer. Advanced hands-on training for distributed and outsourced software engineering. In *ICSE*, 2010.

[9] M. Nordio, R. Mitin, B. Meyer, C. Ghezzi, E. D. Nitto, and G. Tamburelli. The Role of Contracts in Distributed Development. In *SEAFOOD*, 2009.

[10] I. Richardson, A. E. Milewski, N. Mullick, and P. Keil. Distributed development: an education perspective on the global studio project. In *ICSE '06: Proceedings of the 28th international conference on Software engineering*. ACM, 2006.