On the Teaching of Distributed Software Development

Ivica Crnkovic[1], Igor Cavrak[2], Johan Fredriksson[1],
Rikard Land[1], Mario Žagar[2], Mikael Åkerholm[1]

[1]*Mälardalen University, Department of Computer Science and Engineering*
PO Box 883, SE-721 23 Västerås, Sweden
+46 21 {10 70 35, 10 70 35, 15 17 62}
{ivica.crnkovic, johan.fredriksson, rikard.land, mikael.akerholm}@mdh.se

[2]*University of Zagreb, Faculty of Electrical Engineering and Computing*
HR-10000 Zagreb, Croatia
+385 1 6129 861
{igor.cavrak, mario.zagar}@fer.hr

Abstract. *As the software industry moves towards software development projects involving several sites around the world, universities should incorporate this trend into their software engineering curricula. This paper describes the experiences from the development of a university course in distributed software development. Some of the problems of distributed development make it inherently difficult to transfer this domain to the university environment. Also, the concept of "distribution" has penetrated not only the contents of the course but many other levels as well.*

Keywords. Global Software Development, Distributed Software Development, Software Engineering Education.

1. Introduction

Hard competition, a strive for shorter time to market, an increasingly globalized market and internationalized products, and a shortage of software professionals are some of the factors that have lead software enterprises to become globalized [5,15]. Such international software companies devote themselves to mastodon engineering projects as ambitious as did the people of Babel, who started building an enormous tower, supposed to reach unto heaven [1]. Their engineering effort was however spoiled as soon as they started speaking different languages. The software companies of today still face the same challenges as did the inhabitants of Babel, and more: not only are there different languages involved, but also cultural differences, social differences, physical distance, possibly different time zones, different business considerations, etc.

As distributed software development is becoming widespread in practice, and as it meet many problems, universities should incorporate this trend as well into their curriculum [18]. The need for preparing computer science students to the "real world" software engineering problems has already been recognized and is addressed by introducing practical projects and teamwork as a regular part of software engineering courses [8,9,11,12, 13,19]. To our knowledge however, teaching distributed software development at university is very rare, and is restricted either to existing software engineering courses [3] or to case studies and student projects [2]. We have accepted the challenge to develop a university course in distributed software development. Since this subject arguably cannot be realistically taught at one site we are currently developing the course to be simultaneously held in Västerås, Sweden, and Zagreb, Croatia [10]. Throughout the course development, it has been apparent that properties inherent in distributed software development not only pose large challenges to industry, but also make its introduction at university a complex task. In the present paper we describe the issues arising during such a course development.

Section 2 describes course details, section 3 describes how the concept of "distribution" has affected the course on several levels, and section 4 concludes the paper.

2. Course Description

The course is currently under development and will be held for the first time during fall 2003. It will follow the traditional course structure consisting of theory and practice. The theoretical part will be in the form of lectures and self-studies and aims at introducing students to the problems of development distribution and presenting them a roadmap for the practical part of the course. The practical part will be the larger part, and will involve students in one or more distributed projects.

2.1. Course Contents

There are numberous challenges of distributed software development that we intend to teach, and these challenges will be addressed both during the lectures part and the projects part. While they in a sense constrain the course, they are at the same time the aspects we want to teach:

Communication media. The type of communication and the communication media used are crucial issues to be addressed when working in a distributed manner; although meeting in person is invaluable, one has to rely on video or voice conferences, email, collaboration software etc. [5,15]. Students will (most likely) never meet in person due to funding restrictions, but will have to use low-cost communication media extensively.

Configuration management. Although important already in local development, the need for mature configuration and version management of files increase when work is distributed. In the project part, the students will need to share code and documents using configuration and version management tools.

System architecture design. An architecture of a system is not only a result of a technical solution, but it also reflects the structure of the development organization and its development processes. In distributed development the system architecture is an important factor for a successful development process. To understand this the students will have to identify a development process and design an architecture suitable for distributed development.

Formal system specifications. In a distributed development informal information exchange is much more difficult. Further the amount and frequency of the exchange information is limited. For this reason it is important that the system specification is done more accurate and more precisely. This requires better specification of the requirements and the system (in particular the interfaces between the system parts). The aim of the course is to train students in precise specifications, and more formal processing of changes.

Foreign language. When collaborating internationally, there is a language barrier; in the software area the de facto standard language is English. None of the students are native English speaking, but to be able to cooperate they will be forced to use English language both in communication and documentation.

Cultural differences. Cultural distance has been pointed out as one important barrier to overcome in international collaboration [5,6,15]. This includes everything from religion, holidays, and working hours, to cultural “codes” such as whether you should look the person you speak to in the eyes. There are usually different “company cultures” as well (and “university cultures” for that matter). Croatia and Sweden are not too different culturally and people moving between these countries usually adapt very well. Still, there are differences and perhaps prejudices that students will meet.

Synchronous communication. When collaboration is carried out across time zones, the “window of opportunity” of synchronous communication becomes limited or non-existent. Sweden and Croatia are located in the same time zone, but still the ability to communicate synchronously is limited due to flexible working hours. We expect that much of the students will work at home and/or in the evenings, which makes low-cost synchronous communication infrastructure (i.e. fast Internet links) located at the universities inaccessible.

Technologies. The division of work is heavily dependent on an architectural design that allows this, to allow different components to be developed at different sites. It is not yet sure whether the students should design the architecture themselves or be given an architectural design description (to minimize risk). Technologies for distributed applications, if such a product is chosen for the project part, have to be taught as well; this includes e.g. middleware and Internet-based communication.

2.2. The Two Universities

The Department of Computer Science and Engineering at Mälardalen University in Västerås, Sweden, currently offers two undergraduate curricula: one with focus towards programming and algorithms, and the other emphasizing system theory with a traditional hardware/software approach. The research and education at the department is oriented towards industrial engineering, with embedded and real-time systems, and software engineering as two main directions. The students are free to choose courses, but recommendations from the department, requirements for graduation, and relations between courses in practise result in relatively predictable paths of courses. The Swedish can either graduate with a B.Sc. degree after three years, or with a M.Sc. degree after four. The students' knowledge is concentrated on computer related topics, and classical engineering subjects like physics and mechanics are minimized. Most courses contain a practical part and are lab intensive, rather than being theoretically oriented. Teamwork by the means of small project assignments is usually included at the end of each course.

Computing studies at the Faculty of Electrical Engineering and Computing at the University of Zagreb, Croatia, last for nine semesters and aims at the degree Diploma Engineer. The first three semesters are common to both computing and electrical engineering students and consist of classical courses like mathematics, physics and foundations of electrical engineering. There are also some general computing classes such as algorithms and data structures, basic programming, basic computer usage etc. From the fourth semester on the studies become focused on specific courses. For computing, the basic course structure consists of mandatory courses which are progressively replaced by elective courses towards the end of studies, thus allowing slight study profiling. The last semester is dedicated to graduation theses. During the studies students are introduced both to low-level (hardware) and high-level (software) aspects of computer systems. Low-level oriented courses include hardware design and low-level programming; intermediate courses include computer networks, operating systems, etc. High-level courses focus on intelligent systems, databases and programming issues – languages and methods, and more. All courses are lab-intensive and large percent of the final grade is based on the student's lab performance. Two tendencies can be identified in the structure of lab work organization. The first one (and the prevailing one) consists of firmly defined exercises and their results, with individual student work or work in small teams (2-3 students). The second one, more present in elective software-based courses is project work.

The curricula and teaching style are thus similar at the two universities, but some differences are worth pointing out. Both universities hold elective software engineering courses, but while the Swedish students usually attend this course during their third year, the Croatian students attend it during their fourth. The course in distributed software development will be given during the fourth year for Swedish students and the fifth year for the Croatian. The slight difference in educational profile at the two universities implies that division of work can and should be done considering the students' different knowledge. The semester structure and number of parallel courses per semester are different, so there has been a certain amount of "puzzling" to make the course fit into both universities' semester structure. The start and end dates must be coordinated, as well as holidays. There are also differences in availability of e.g. lab rooms; at Mälardalen University there are more resources per student, which are available until late in the evening.

2.3. Theoretical Part

The contents of the theoretical part have several aims:

- First, to make the students aware of driving forces behind work distribution, through means of theoretical principles used in different domains of software engineering and from examples from software industry.
- Second, to describe the challenges involved, and what methods and tools can be used to alleviate distance problems (as was described in section 2.1).
- Third, to prepare them for the project part, by introducing the tools to use, and the assignment.

Although the previous knowledge cannot be assumed to be identical when the course starts, the theoretical part of the course can indeed be made identical. At the very least, the same literature and lecture slides can be used. But there is also the opportunity to fully make even this part of the course distributed: students can attend lectures held at the other site by means of distance learning tools. Both universities involved are actively pursuing distance learning for domestic students, so this would be a natural continuation of that direction.

2.4. Practical Part

The practical part of the course will be in the form of a project aiming at developing a software product, as is common in many software engineering courses [8,9,11,12,13,19]. Since we intend to teach distributed software development it is important that the students already are familiar with e.g. project planning and configuration management. We want the problems the students will face to rather be related to the distributed project work.

This part of the course will be focused on analyzing, designing, developing and testing of one or several projects ordered by a "customer" and will last about seven weeks. Members of the staff on one or both universities will play the role of the customer. As most software efforts of today involves such activities as maintaining, modifying, or integrating legacy systems, using an API (Application Programmer's Interface), and extending or reusing (parts of) existing software, we believe an "advanced" software engineering course such as this should introduce these elements as well. We therefore intend the projects to extend or integrate existing software to increase their value. When choosing application domain, it is important that the source code is legally available, or that there is an API (not necessarily well-documented, since we want to give the students a realistic experience). We are currently discussing building applications on top of, possibly even integrate, the following software: Bugzilla [4], CVS (Concurrent Versions System) [7], ICQ [14], Microsoft NetMeeting [16], and MSN Messenger [17] . The tools will thus be within the domain of distributed collaboration. One way to extend them is to store communication sessions in some sort of context, thus providing project traceability and visibility; another is to build a tool that uses CVS data to analyze project work (e.g. who did what and when).

To give the students a sense of the problems they may run into during the actual project, they will be encouraged to spend some time learning about "the other" country – Croatian students will study Sweden and vice versa. Also, depending on the project they are allocated to, students will have the time to familiarize themselves with required technologies. Self-study and/or small task-force teams will be the primary organizational units during this phase which is not expected to last more than 2-3 weeks, depending on the site and the time available.

At the end of the project, the students will present not only their products, but also analyze the project work. What problems did they have? How did they solve them? These experiences will be forwarded to next year's course. Product requirements will also be continuously "inherited" by next year's course. The students' analysis will be used as a basis for requirements. In this way requirements and resulting products are "bootstrapped" along subsequent courses (and possibly graduation theses).

3. Distribution

In this section we elaborate on how the subject "distributed software development" has affected not only the contents of the course, but how the concept of "distribution" has penetrated many other aspects of the course as well. Some consequences have been inevitable while others have been consciously included for educational reasons. Each of the five "levels of distribution" is discussed below.

First, the theoretical part is rather simple: the challenges and solutions to efficient distributed software development are presented to the students in the familiar lecture form.

Second, to give the students a realistic practical experience, practical collaboration between two parties is necessary. Involving an industrial counterpart makes focus shift from project to product, from education to end product. Although other courses in distributed software development have employed real-world customers [2,3], we believe that in the long term it is essential that externally managed resources do not present a too high risk to the course itself, which is why we have chosen not to involve industrial partners. Although we want to teach distributed software development in as realistic environment as possible, there are limitations on what is possible. As we have chosen not to involve an industrial counterpart, the most obvious drawback is the amount of funding. The use of mid- and high-cost methods such as telephone-conferencing and travel [2,3], although found unavoidable in "real-world" projects, will therefore be limited. Another difference is that there is neither the option of assigning more people, nor can the delivery date be late; we therefore have to minimize risks by e.g. giving them reasonable requirements and monitor their architectural design carefully (or perform this design ourselves). We also run the risk of students drop out in the middle of the course, jeopardizing project completion.

Third, the problem domain for the software to be developed in the practical part was chosen to be distributed collaboration as well. The tools developed will be used in the course to the greatest extent possible; it will e.g. be possible to use a "project analyzer" built on top of CVS to analyze the project building this tool. Since the students use the tools themselves, their evaluation of the products at the end of the course will be realistic and more valuable than if the product was intended for virtual users. We plan to make the products available to the public community, thus employing a potentially large number of users in testing the product usability and quality. This will provide feedback used as part of the requirements for the next course. Maintenance and refinement of the products will be (at least partially) ensured by assigning product-related graduation thesis to some of the students participating in the project.

Fourth, the products might, or might not, be implemented as distributed systems, e.g. in the client-server style or in more sophisticated configurations. Distributed systems and distributed development both rely heavily on a successful architecture. However, as said above, all projects where components dependent on each other are developed at different sites run a considerable risk, whether or not they will execute at different nodes.

Fifth, since the course will involve two universities, the actual development of the course has to be carried out in a distributed manner. One of the Swedish course teachers have visited Zagreb for six months to develop the course jointly with one of the Croatian teachers, forming a "bridgehead" [6] between the two sites helping to alleviate organizational and cultural problems that arise along the way. The course teachers in Sweden have been involved through videoconferences, email, and document sharing. This setting enabled the teachers to get "first-hand experience" of the problems involved in distributed *course* development, if not distributed software development. Some problems experienced have been of a technical nature (such as poor sound quality during video conferences), others of a more personal character (such as discussing different people's ideas and distributing work). These experiences are invaluable when it comes to teaching this subject.

4. Conclusion

Simulating the real world at university is a challenging task. The more complex phenomena to be taught, the greater the challenge. In this paper, we have presented the challenges encountered during development of a university course aimed at teaching distributed software development. To provide a realistic environment, it is not sufficient to develop a course locally at one university. Either should two (or more) universities be involved, or a university and a remote industrial partner. In a university environment, one has to tradeoff the resources made available by involving an industrial partner for the risk in terms of focus shift towards product instead of project and education.

The "distributed" aspect of the course affects the course development on several different levels, in some ways unavoidable, in some ways as a conscious decision. Apart from teaching the subject theoretically, the students will work practically in as realistic setting as can be provided. To increase the educational value, the product or products to be developed during the practical part of the course are tools used for distributed work. The tools may be implemented as distributed systems. And finally, the course development itself had to be carried out in a distributed manner, since there are two counterparts (in our case two universities) involved.

We hope that our effort of teaching students how to face the challenges of distributed software development will equip them with knowledge sufficient to successfully take part in or lead distributed projects, both us and them being able to handle the challenges of physical and cultural distance better than the people of Babel did.

5. References

[1] Book of Genesis, Chapter 11, verses 1-9

[2] Brereton P., Lees S., Bedson R., Boldyreff C., Drummond S., Layzell P., Macaulay L., Young R., "Student Collaboration across universities: A Case Study in Software Engineering", Proceedings of 13th

Conference on Software Engineering Education and Training (CSEE&T), IEEE, 2000.

[3] Bruegge, B., Dutoit, A.H., Kobylinski, R. and Teubner, G. "Transatlantic project course in a university environment", Proceedings of 7th Asia-Pacific Software Engineering Conference (APSEC), 2000.

[4] "Bugzilla Project Home Page", web page, URL: http://www.bugzilla.org/

[5] Carmel E., "Global Software Teams: Collaborating Across Borders and Time Zones", ISBN 0-1392-4218-X, Prentice-Hall, Upper Saddle River, NJ, 1998.

[6] Carmel E., Agarwal R., "Tactical Approaches for Alleviating Distance in Global Software Development", IEEE Software, volume 18, issue 2 , IEEE, 2001

[7] "Concurrent Versions System", web page, URL: http://www.cvshome.org/

[8] Crnkovic I., Larsson M., and Lüders F., "Implementation of a Software Engineering Course for Computer Science Students", Proceedings of 7th Asia-Pacific Software Engineering Conference (APSEC), 2000.

[9] Crnkovic I., Land R., and Sjögren A., "Is Software Engineering Training Enough for Software Engineers?", Proceedings of 16th Conference on Software Engineering Education and Training (CSEE&T), IEEE, 2003.

[10] Cavrak I., Land R., "Taking Global Software Development from Industry to University and Back Again", Proceedings of ICSE International Workshop on Global Software Development (GSD), 2003.

[11] Daniels M., Faulkner X., and Newman I., "Open ended group projects, motivating students and preparing them for the 'real world'", Proceedings of 15th Conference on Software Engineering Education and Training (CSEE&T), IEEE, 2002.

[12] Dawson R.J., Newsham R. W., and Fernley B. W., "Bringing the 'real world' of software engineering to university undergraduate courses", IEE Proceedings In Software Engineering, volume 144, issue 5, 1997.

[13] Dawson R., "Twenty Dirty Tricks to Train Software Engineers", Proceedings of 22nd International Conference on Software Engineering (ICSE), ACM, 2000.

[14] "ICQ", web page, URL: http://web.icq.com/

[15] Karolak D., "Global Software Development: Managing Virtual Teams and Environments", ISBN 0-8186-8701-0, IEEE Computer Society Press, Los Alamitos, CA, 1998.

[16] "NetMeeting Home", web page, URL: http://www.microsoft.com/windows/netmeeting/

[17] "MSN Messenger", web page, URL: http://messenger.msn.com/

[18] Shaw M., "Software Engineering Education: A Roadmap", Proceedings of the 22nd International Conference on Software Engineering, ACM Press, New York, NY, 2000.

[19] Wohlin C. and Regnell B., "Achieving industrial relevance in software engineering education", Proceedings of 12th Conference on Software Engineering Education and Training (CSEE&T), IEEE, 1999.