Parallel Software Engineering Student Projects

Juha Taina
University of Helsinki, P.O. Box 68, FIN-00014 UNIVERSITY OF HELSINKI
taina@cs.helsinki.fi

Kimmo Simola[1]

Abstract –
Software engineering education requires a practical course where students can learn processes and practices in a controlled environment. Since software development is usually done in teams, such a course should be a team work course. Due to this, we have a one-semester software engineering project where five to seven students form a team. Each team has an assigned customer who needs a software product. Next to traditional one-team projects, we have parallel team projects where several teams have the same customer and problem scope. The teams work separately and optimally they do not have any cooperation. Each team has its own product to develop from equal specifications. As a result, the customer gets several products from the same original specification. After three years of parallel projects, the resulting software has been unique enough for various comparisons. In this paper, we summarize our experiences with parallel teams and draw conclusions of their advantages and disadvantages.

Index Terms – Software engineering education, Student team work, Parallel project, Empirical software engineering

INTRODUCTION

There is a clear need for a practical course in software engineering education. Not only is such a course useful in learning important software engineering skills in practice, but it is also useful in empirical software engineering research. Indeed, one of the most interesting software engineering education research fields is to summarize experiences of student software engineering projects.

The structure of the course may vary from school to school, but most schools have a student software engineering project. For instance, Dawson lists in his article twenty tricks to use in software engineering course training [4]. Alfonso and Mora have reported their experiences in software engineering group work [1], Brazier et al. have written an article about their software engineering student projects in Brazil [2] and Coppit and Haddox-Schatz report how their large team projects work in software engineering courses [3]. Pletch and Agajanian report a project that closely resembles to real-world software projects [5]. All these and many others have been reported in recent literature. Student software engineering team work is a core course in a software engineering education of any level.

At the University of Helsinki, Department of Computer Science, we have a software engineering project course where students are able to get a feel of software development in a safe environment. It is not only students that benefit from the course, since the projects have customers with real software needs. We may also conduct experiments at the Department using the project teams. There are more opportunities to conduct experiments than there are feasible experiments, since there are about 20 project teams every year (some are even held during a summer semester).

Using software engineering teams in experiments is common in empirical software engineering research. While students are not professional software engineers, they are not total amateurs either. As Tichy has mentioned, students can be used in experiments when certain conditions are met [7]. We have found that student software engineering projects are a very good platform for such experiments. This was our main goal when we first started to use parallel projects where several teams work individually on the same software problem.

In this paper, we give an overview of the software engineering project course and discuss several approaches to doing research using the project teams. Along with the experiment settings, we also summarize the results of the experiments we have conducted so far.

The rest of the paper is organized as follows. We will discuss the software engineering projects of the Department concisely. This information serves as a base for a description of parallel project setting used to conduct research. After that, we will discuss the actual experiments we have conducted so far and reflect upon the results of these experiments. Finally, we will analyze the lessons we have learned during the parallel projects.

SOFTWARE ENGINEERING PROJECTS IN UNIVERSITY OF HELSINKI

The software engineering project is a mandatory course for all computer science majors at the University of Helsinki. It is often the last course before the Bachelor's thesis. It lasts one full semester and requires 17-20 hours of work from each participant every week. There are about seven project groups every semester and each of them consists of 5-7 students and an instructor.

The projects follow either a linear or an iterative process model. The iterative model may have 2-3 short iterations. The process model and number of iterations are a decision made by the project group but we remind the students that

[1] Kimmo Simola, University of Helsinki, kimmo.simola@cs.helsinki.fi

extra iterations also imply extra work. In both models, the amount and quality of documentation is expected to be equal. Because the project is still a study module, the development cycle ends after software delivery and system deployment.

Project groups are assembled using information from course registrations. Students answer several questions in a registration form ranging from their preferred project topics to their interests in software engineering. The questions are kept simple to make the enrollment process smooth. As a result, the groups are highly heterogeneous. Some participants may be beginners in project work while some might be experts.

The students get the required theoretical knowledge from previous courses and in some cases from work experience as well. The project itself is a practical course. Although each group has an instructor who is an expert on software engineering, it is the students' contribution that matters.

The customers have a real need for the solution systems to be developed during the projects. There are no predefined standard topics for the projects although some topics may accidentally resemble others. A typical customer is a researcher, a teacher, an administrative person, or even someone from outside the Department.

PARALLEL PROJECT BACKGROUND

We started our parallel team projects in the spring semester 2004. Since the software engineering project is mandatory for all our students, we wanted to let as many students as possible to participate in a project they were most interested in. Most of our project teams have individual work topics, as there are more available topics than teams. Since the topics come from real customers, some of them may gather more popularity among students than others.

Several teams from the same problem scope give us more freedom when selecting students to teams. For instance, when students sign in to the course, we ask about their working habits. We can use this information to create teams that have similar schedules.

Parallel projects have the same problem description and the same stakeholders. The teams themselves are independent of each other and are not supposed to be in contact with each other. However, it is hard to ensure that students do not communicate with each other if they know each other well.

The parallel project setting gives us a chance to conduct controlled experiments using the project teams. We may allocate students to parallel teams based on some research criteria, such as gender or age. On the other hand, we may instruct one of the teams to use different techniques or processes while other parallel teams act as control groups.

One unfortunate disadvantage of the parallel projects is that even with good teams only one final product is selected for future development. While we do not tell the students which team was better, they can easily guess it from customer satisfaction and future development plans. Yet several teams have reported that the other team in a parallel project did not affect their work at all.

PARALLEL PROJECTS

Our first parallel project semester was also the most productive one when counting the number of parallel projects. We had three regular parallel projects where two teams worked on the same problem: a project for science magazine referees, a project of a sea eagle observation management system, and a project of a machine language simulator. Next to these, we had a cooperated project with the University of Petrozavodsk, Russia, where two teams worked separately on different issues of the same software problem at different geographical locations [9].

In the summer semester 2004, we had a parallel project about an office room management system. Unfortunately also in the summer semester 2004, our department moved to our current location and due to this we had only short projects that summer. In the short projects the participants were asked to work for 30 hours a week. This turned out to be too much. While the total hours of the summer teams were close to 240 hours, the tight schedule did not allow the project participants to have time to process their ideas. As a result, the summer projects had severe difficulties with schedule and software functionality.

In the autumn semester 2004, we had a parallel project of a taxi system simulator. This parallel project went well, but unfortunately we do not have statistics left about it. At that time, we were still in the middle of our moving and hence not all our systems were fully functional.

In the spring semester 2005, we had a parallel project of a participating student profiler system. This was an interesting project because its customers looked at the problem from different angles and the teams also had very different types of solutions.

In the summer semester 2005, we had two parallel projects: a project of a generic drawing application and a project of an assistant teacher meeting scheduler. This semester we got heavy variation in project lengths and quality. Neither parallel project was a success, but we learned valuable lessons about team structure and customer participation from both of them.

Our latest finished parallel project was in the autumn semester 2005 about a room reservation system for a student organization. This project was a success. It was also a part of our experiment of female-only software engineering project teams [6].

After the autumn semester 2005, we had 1.5 years without parallel projects. Finally, in the ongoing semester (spring, 2007) we have a parallel project about a meeting reservation system for teacher tutors. Since the project is not finished yet, we do not include it in this summary.

So far we have had total ten parallel projects, eight of which are listed in this paper. Of those projects, we have used three in empirical software engineering research: the sea eagle observation management system project, the data communication protocol through animation project, and the room reservation system for a student organization project. Nevertheless the other parallel projects have been useful also from the education research point of view. While not all our parallel projects had research interests, they still have offered

us many valuable lessons. Thus, it is appropriate to summarize the projects here.

I Nordic Journal of Computing referee system

The Nordic Journal of Computing (NJC) science magazine is published in our University. In the spring semester 2004, we had a project where the managers of the magazine wanted to have an article evaluation system for anonymous referees. The system kept track of received articles, referees, referenced articles, referee comments, and published articles, among other things. As such it was specified to be a complete article control system for a small or medium magazine using anonymous referees.

The parallel project had two teams: NJC1 and NJC2. The members of NJC1 were selected to have experience in user interface design and digital media courses. The members of NJC2 were not as experienced in that field, but also some of them had some user interface and digital media background. While we were interested to see whether a user interface background would affect the result, we did not have a real controlled experiment due to the user interface skills of the NJC2 members.

Both teams created good software on schedule. We noticed that the NJC1 team indeed created a product that was more intuitive and easier to use than the NJC2 product. However, this analysis was based on our subjective opinions. From a user interface research point of view, the differences were small. It is also not clear whether the students' background affected the result or was it because the teaching assistant of NJC1 had strong interest in user interfaces.

Next to the user interface issues, we were interested to see how well our new parallel project paradigm would work. In this project, it worked very well. Both teams had the same problem scope but already starting from the requirements analysis they worked separately from each other. Also the customers of both teams were able to keep the teams separate. We got two products that had the same background but which looked and felt different from each other.

II Sea eagle observation management system

The sea eagle observation management system is a member of a large product family that has been under development in our software engineering projects for the last 15 years. This time the teams were asked to create software for sea eagle observers. The software was specified to allow observers to input data to the system from forms that they filled on the field near sea eagle nests. The system should also output various reports about input data.

In the spring semester 2004, we started a case study where we would have a female-only and a male-only team working on a parallel project. We made the case study in two semesters and in the spring semester 2004 we chose the sea eagle observation management system to be the first project to observe.

This parallel project had two teams: Kotkat (Eagles in English) with six female students and Hali2 (Hug2 in English) with five male students. Members of both teams were selected according to their preferences but since we do not get that many women to our projects, we basically had to gather all possible female participants that had at least a small interest to this kind of a project.

In this project, both teams had the same two customers. This was not a good idea since the customers often argued with each other about issues in front of the students. As a result, the students were confused of what they should do and whose word to follow. Fortunately, the project that had a rocky start ended relatively well. The members of the female-only team were especially happy with the project and the members of the male-only team did not complain either.

III Machine language simulator

The machine language simulator was designed for our course about computer organization. It was specified to emulate a virtual processor with its own machine language, registers, small memory, stack, and I/O-operations. The system would execute students' machine language code and show the results in registers and memory.

In this project, we had two teams: Koski (Rapids in English, it is also an acronym of the Finnish word "Konekielisimulaattori" – a machine language simulator) and Malan (an acronym from a Machine language simulator). The participants of the teams were randomly selected from interested students. We did not set a research goal to the teams.

Although this parallel project did not have a research theme, it turned out to be an interesting one. In the project, the customer did not completely distinguish the teams from each other. At certain times, the customer even encouraged the teams to co-operate. This confused the teams and no doubt affected the results. It also caused the resulting products to noticeably resemble each other.

IV Student registration profiler

Our next parallel project was in the spring semester 2005. This time the teams were asked to create student registration profiler software. Such software would allow the supervisors of courses with small group work to decide whether a student would be allowed to attend the course, and if yes, which small group he or she would participate. For instance, we use a later version of this software in our software engineering group work when we select students to project teams.

In this parallel project, we had two teams. The teams were Ilpo (a male Finnish name) and Proffa (a Finnish slang word for a professor). Ilpo had five students and Proffa six. Both teams had their own customer and, fortunately for us, both customers had a very different idea of what functionality the resulting software would have and what kind of a user interface it would have.

We chose the members of the teams randomly. Both teams were about equal in size and skills. The teams did not have much contact with each other and the customers were able to keep the teams separate from each other. The biggest differences were in requirements and it clearly affected the products.

As a result of the problem and customer preferences, we got two pieces of software that answered the original problem definition but that were otherwise quite different from each

other. The teams had chosen completely different approaches which showed well in the final products.

V Generic drawing application

In the summer semester 2005, we had two parallel projects. The first one was a generic drawing application. The application requirements stated that the application has to support different types of modeling languages. It was defined to be a core framework for a family of different modeling software.

The summer projects did not manage to create suitable core software. We had teams Canvas (the same in English) and Oops (the same in English). The Oops team at least tried to create good software and their resulting product was later expanded in a new single team software engineering project. The Canvas project, on the other hand, failed miserably.

While the Canvas project was a failure from a product point of view, we learned a valuable lesson from it. First, the size of the Canvas team eventually became too small. It had originally five participants, but two of them cancelled the project. Second, the project had several students whose mother tongue was neither Finnish nor English. A language barrier, early cancellations, and general lack of interest to the project were the main reasons for the failure of this project. None of the issues alone would have destroyed the project, but together they were too much.

VI Assistant teacher meeting scheduler

Also in the summer semester 2005, we had a parallel project of an assistant teacher meeting scheduler. The customer wanted software that assistant teachers could use in laboratory courses. The purpose of the software was to allow students to select from a set of predefined time slots when they would like to meet their teaching assistant.

In this project, we had teams Aija (a Finnish female name) and Sahara (the desert in North Africa). The Aija team had five members and the Sahara team had four members. We selected the members of the teams from emails because originally we had too few project proposals and we asked one of our colleagues to create a new proposal. The new proposal turned out to be very interesting to the students. We found enough volunteers for two parallel teams.

This project had some very good characteristics and some very bad ones. A positive thing of the project was that the customer was able to keep the teams separate, and the teams had very strong ambition to finish the project. A negative thing of the project was that the customer had unrealistic requirements which the teams did not prioritize well. As a result, especially members in the Aija team worked extra hours in order to finish the project on schedule.

VII Room reservation system for a students' association

We concluded our spring semester 2004 female-only team experiment with a new parallel team experiment in the autumn semester 2005. Again, we had a female-only team and a male-only team, but unlike the previous time, this time we had two customers who had their own teams. This time the project proposal specified software that members of a students' association could use to reserve meeting rooms, sauna, and other common facilities of the association.

This time we had teams Potta (a Finnish slang name for the students' association that wanted the software – the word is also a potty in Finnish) and Innova (comes from innovation). The Potta team had five female students and the Innova team had six male students. We wanted the Potta team to have six female students but one student cancelled her participation before the project had started.

The idea of having separate customers worked well. This time the teams were totally independent of each other. The results were interesting, too. Both teams did a very good job and the quality of both products was about equal.

After the teams had finished, we also finished our case study about female-only software engineering teams. We found out that female-only teams work equally well as male-only or mixed teams, but perhaps the members of female-only teams are more social and their working habits are somewhat different. We have published a complete description of the case study in a separate paper [6].

VIII Data Communication Protocol through Animation

While in all the previous projects the teams worked (or at least tried to work) separately from each other, our data communication protocol through animation project was different. This time both teams worked on different aspects of the same problem. The interesting part of the experiment was that the teams were geographically distributed.

The team at the University of Helsinki (Dacopan UH) had three Finnish students, two Spaniards, and one Finnish-American. The team at the University of Petrozavodsk (Dacopan UP) had five Russian students. As far as we know, this is the most international geographically distributed student team project reported in literature. However, although Dacopan UH team did not have a common language, it did not ruin the project. Instead, multi-language students gave new views to the problem and its solutions.

In the project, we considered it wise to start the project with a common kick-off period. This was arranged in Helsinki, where both teams worked together with the customer. In three weeks, the teams and the customers found most of the requirements of the software. A second joint working period was arranged towards the end of the project in Petrozavodsk for a period of ten days. The period was used for joint integration testing and software demonstration. However, part of the scheduled time was actually used for coding since the project was a little late.

Our experience with the Dacopan project was positive. In spite of the differences in team members' backgrounds and little of face-to-face communication, the students in both teams produced a high-quality product within the given schedule. The feedback from the students was positive and they all agreed that these types of projects are useful. The Dacopan project participated later in the Microsoft technologies in software engineering and software development contents where it won the first place. Our cross-cultural software engineering project experiment is fully described in our report [8] and it is summarized in a later conference article [9].

LESSONS LEARNED

Most of our parallel projects have been educational successes. We have noticed that a good parallel project where two or more teams work on the same problem definition has the following characteristics:

- All teams are about equal in size and skills.
- All teams have their own customer and the customers of the teams do not know about the advances of the other team.
- The original problem definition is the same with all teams but the customers of the teams have slightly different preferences: for instance they may represent different stakeholders or have different interests in human computer interface issues.
- The teams are neither asked to compete with each other nor do they get the impression that they should do it.
- The teams do not need any information from the other teams. All needed information comes from the customer of the team.
- If the teams have the same customer, the customer will not let his or her biases to affect his or her relationship to the teams.

It is important that the teams are about equal in size and skills. If the sizes of the teams are very different, as happened in our Canvas project, the created products are not comparable. Also since both teams are aware of each other's work – we have noticed that this is unavoidable – the knowledge of smaller available resources often makes the members of the smaller team to work too hard or give up project work almost completely.

Unfortunately, it is not possible to completely avoid teams of different sizes. In our case, in the Canvas project we had five participants. Unfortunately one student cancelled the course very early and another student cancelled it after about one third of the project was over. The other three were never very interested in the team work and after the cancellations their work motivation dropped more. The result of the project was barely acceptable.

The equal skill factor is important as well, but it appears that it is not as important as the size factor. We have had teams where the participants have had quite limited skills and yet they have got reasonably good results. We believe that the feeling that the team members work together for a joint goal and can trust each other is far more important than individual skills. We have noticed that the best chances for this to happen are when a team has 5-6 students. Members of smaller teams may try too hard or too little. Members of larger teams seem to somewhat miss the feeling of being a uniform team.

We have noticed that the best way to jeopardize a parallel project is to let the teams have the same customer. Few customers are able to completely distinguish their opinions and knowledge of other teams. This extra knowledge percolates in team meetings especially in the requirements analysis phase. It also shows in the relationships between the customer and the teams since most customers soon find out which team is their favorite one.

On the other hand, the best way to have a good parallel project is to let the teams have customers who have individual preferences within the problem scope. Unfortunately, we have had only three such projects so far: the Nordic Journal of Computing referee system project, the student registration profiler project, and the room reservation system project.

One of the biggest issues in our parallel projects is that parallel teams tend to compete with each other. This is especially true when we have an experiment where teams have easily recognizable differences. For instance, in our female-team study, both times the members of both teams of course immediately realized that team members were not randomly selected. This kind of a competition situation is bad for education, research, and team work. When teams compete with each other, they tend to forget the basics of software engineering. They want to have as much functionality in a limited time as possible. As a result, quite often they create enormous software that is badly designed and tested.

While it is not possible to forbid students from competing with each other, the customers should be wise enough not to let their feelings and preferences affect team work. Alas, this is not always the case. We have had customers who were not able to keep their teams separate. In the weakest form, the customer lets his or her knowledge of one team's work affect the dialogue with the other team. For instance, quite often one team finds out an elegant solution to some functionality issue of the developed software. If their customer is not careful, he or she may easily hint the other team about the solution. In the strongest form, the customer forces all teams to use a solution that is introduced by one of the teams.

A summary of the parallel projects as listed in this paper is in Table I. The table lists the team names, the number of students of a team, the total hours, the minimum working hours, the maximum working hours, and the average working hours. The students of the projects account their private working hours and tasks into our metrics system. The hours spent in a project do not directly affect grading and we have told this to the students as well. As such, we trust that the listed hours are fairly realistic with natural variation.

As we can see from Table I, the average working hours are about the same in all projects. The only clear difference is in project Canvas. That project was a most unfortunate one and it also shows in its hours. Next to the Canvas project, we have fairly equal projects in average hours. The largest differences are in the Aija project (310.2 hours) and Hali2 project (200.7 hours). The Aija project suffered from unrealistic requirements, as listed earlier. The low hours of the Hali2 project are somewhat a mystery to us. The team did a good job so low hours do not show that much in quality. The most probable reason for their low hours is that they did not really keep track of all their working hours. Sometimes our students consider this kind of work waste of time since it does not directly affect course grades.

While the minimum hours and maximum hours vary a formidable amount from project to project, it is not a serious issue. When 5-6 students join a team without prior

knowledge of each other, it is not a surprise that someone will do as little as possible in the team. As long as he or she does not encourage others to slip their duties, it does not affect the team that much.

TABLE I
PARALLEL PROJECT STATISTICS

Name	#	Hrs tot.	Hrs min	Hrs max	Hrs avg
NJC1	6	1488	218	292	248,0
NJC2	6	1541	247	270	256,8
Kotkat	6	1551	245	274	258,5
Hali2	6	1204	175	242	200,7
Koski	6	1444	224	260	240,7
Malan	5	1120	144	265	224,0
Ilpo	6	1429	190	269	238,2
Proffa	5	1183	224	253	236,5
Canvas	3	362	113	128	120,5
Oops	6	1662	243	313	277,0
Aija	5	1551	229	367	310,2
Sahara	4	1093	222	324	273,3
Potta	5	1372	246	298	274,3
Innova	6	1526	193	249	254,3
Dacopan UH	6	1787	257	332	297,8
Dacopan UP	5	1211	167	296	242,2

A graphical summary of the Table I data is in Figure 1. The figure shows the minimum, average, and maximum hours of each team in each parallel project as listed in the previous section. The closer the different dots of a project are, the more equally the members of that team have worked. The closer the dots are to the 240-hour line, the closer the project has been of the recommended size. Again, the figure shows that most teams have had about the right project effort and that all members of a team have worked about as much.

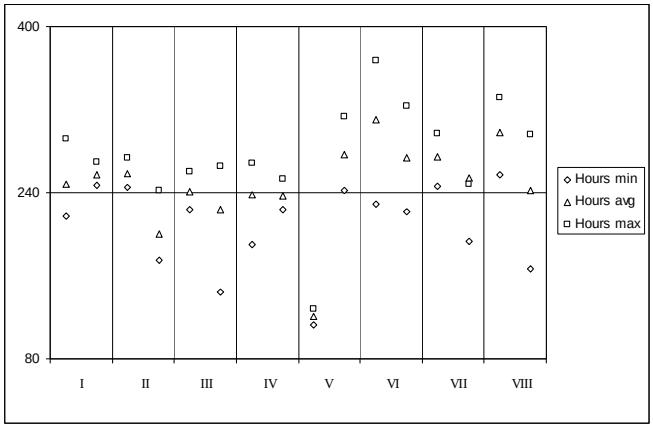

FIGURE 1
PARALLEL PROJECT HOURS.

CONCLUSION

Our experiences with parallel software engineering student projects have mostly been positive. Most teams have not complained about the artificial situation of two parallel teams. Usually at the beginning of the project teams think that they must compete with each other or at least find out what the other team is doing, but when time passes and software requirements are gathered teams seem to forget that the other team existed.

The best of our parallel projects have been good for students, customers, and researchers. We have had several such projects. A common thing to the projects is that their participants have really wanted to join the teams, each team has had its own customer, and teams have been equal in size and strength. We think that these three issues are almost mandatory for a successful parallel project.

In our three years of parallel projects, we have learned that such projects are a good research resource. Having good experiment conditions with parallel projects is in fact a smaller problem than finding out a good research topic to experiment with a parallel project. We expect to have several parallel project experiments in the future. We hope that this paper will encourage other schools to try them as well.

ACKNOWLEDGMENT

This work was supported by the University of Helsinki, Department of Computer Science. We wish to thank Prof. Inkeri Verkamo and two anonymous reviewers for their valuable comments and suggestions.

REFERENCES

[1] Alfonso, M. I. and Mora, F., "Learning software engineering with group work", *Proceedings of the 16th Conference on Software Engineering Education & Training, 2003 (SEE&T 2003)*, March 2003, pp. 309 – 316.

[2] Brazier, P., Villalobos, M. C., Taylor, M. B., Basu, K. and Sircar, T., "A Cross-disciplinary Software Engineering Project Implemented as a Web Service", *Proceedings of the International Conference on Engineering Education 2006 (ICEE'06)*, July 2006 (in CD-ROM).

[3] Coppit D. and Haddox-Schatz, J. M., "Large team projects in software engineering courses", ACM SIGCSE Bulletin, 37, 1, January 2005, pp. 137 – 141.

[4] Dawson, R., "Twenty Dirty Tricks to Train Software Engineers", *Proceedings of the 2000 International Conference on Software Engineering*, June 2000, pp. 209 – 218.

[5] Pletch, A. and Agajanian, A., "A software engineering project that looks like the real world", *Journal of Computing Sciences in Colleges*, 22, 6, June 2007, pp. 92 – 99.

[6] Taina, J., "Female-only Student Software Engineering Teams – a Case Study", *Proceedings of the International Conference on Engineering Education 2006 (ICEE'06)*, July 2006 (in CD-ROM).

[7] Tichy, W. F., "Hints for Reviewing Empirical Work in Software Engineering", *Empirical Software Engineering*, 6, 4, December 2000, pp. 309 – 312.

[8] Verkamo, A. I., Taina, J., Bogoyavlenskiy, Y., Korzun, D. and Tuohiniemi, T., "Experience in a distributed cross-cultural student software project", *Report C-2004-65*, University of Helsinki, Department of Computer Science, 2004.

[9] Verkamo, A. I., Taina, J., Bogoyavlenskiy, Y., Korzun, D. and Tuohiniemi, T., "Distributed Cross-cultural Student Software Project – a Case Study", *Proceedings of the 18th Conference on Software Engineering Education & Training, 2005 (SEE&T 2005*, April 2005, pp. 207 – 214.