Designing a Multi-disciplinary Software Engineering Project

Patricia Lago, Joost Schalken and Hans van Vliet
VU University Amsterdam, the Netherlands

E-mail: [patricia, joost, hans]@cs.vu.nl

Abstract

Software engineering courses often include some form of project, aimed at bridging the theory-practice gap. These projects tend to emphasize technical topics. Because software engineering has an important organizational and social dimension as well, and because software engineering courses may be taken by non-CS majors too, there is every reason to also include these non-technical issues in the software project. In this paper we discuss how we designed a multi-disciplinary software engineering project course that pays attention to non-technical issues. We report about our experience in four editions of the same project, and discuss lessons learned. Our general conclusion is that software engineering projects need to have specific characteristics to attract a multi-disciplinary student population.

1 Introduction

Software engineering is not an easy subject to teach. It easily remains very abstract, it requires some experience in the field to be fully understood and appreciated, and it involves a certain degree of creativity. Most software engineering programmes try to fill this theory-practice gap by means of student projects in which the theory is put into practice. There are many forms such a project can take, varying from projects based on real industry examples [5] to ones that introduce obstacles and dirty tricks into student exercises [4]. Software engineering combines technical and social skills, as well as collaboration among people with different backgrounds and skills. This is more and more the case in present-day software engineering, where global development and outsourcing have become popular. This adds a further difficulty to teaching realistic software engineering, as well as setting up the corresponding projects.

If the students share the same CS background, these projects already present a technical challenge. If the students also have different, multi-disciplinary backgrounds, we need to organize these projects even more carefully. The risk is that we fail to involve the students that misunderstand the goals, and will never appreciate either the technical or the more social aspects of software engineering. In our university we have the opportunity to experience such a multi-disciplinary environment: the students attending our software engineering project course come from five different specializations, and hold different skills, backgrounds and education interests. At the same time, this multi-disciplinary environment presents a challenge: how to set up a project exploiting at best such a rich environment and involve all types of students?

In this paper we report on our experiences in designing a large software engineering project course. Its aim is to have students appreciate and recognize a number of relevant aspects of software development. Based on the lessons learned during the 2003 and 2004 editions, we redesigned the project to specifically attract and trigger the interest of a multi-disciplinary student population. In the 2005 and 2006 editions we gathered feedback from the students, and observed the extent to which our objectives were achieved. In this paper, we reflect on what did work as planned and what did not work, and why.

In redesigning the course, we hoped to increase the involvement of non-technical students. We partly succeeded. The social, cognitive elements from the improvements have had an impact, whereas the business/organizational aspects have had a minor effect, if at all. This impact was significant as far as grades are concerned. As far as student perception is concerned, the software engineering project still appeals less to students with a social background.

978-0-7695-3539-5/09 $25.00 © 2009 IEEE
DOI 10.1109/CSEET.2009.42

2 Background

Like most Dutch universities, ours does not offer separate computer science (CS) and software engineering (SE) degrees. We have a three-year bachelor's program and a two-year master's program in CS. Most students enroll in the bachelor's program right after high school. The program doesn't have much specialization and has one general SE course. This course includes both theory and project work. The master's program contains a series of more specialized SE courses.

The VU University rates its SE course's theoretical and practical parts at 4 and 8 ECTS credits, respectively. (In the European Credit Transfer System, 1 ECTS amounts to approximately 28 study hours; a full year is 60 ECTS.) The course lasts 12 weeks. Students are scheduled to take it in the second year of their bachelor's program, which means they have little maturity in CS or SE when they enroll. The course is compulsory for students in CS, Artificial Intelligence (AI), Multimedia and Culture (MMC), Business Informatics (BI), and Business Mathematics and Informatics (BMI). Typically, around 100 students enroll each year.

2.1 Organization of the SE project

The project SE (PSE) lasts twelve weeks. It is split into two RAD (Rapid Application Development, [17]) iterations (see figure 1). At the beginning of the project, the students are given an assignment. This is different every year. The assignment describes in very general terms the type of software application that must be developed, its quality, the technology and the minimal functional requirements. No external organization is involved in the project. The project is on average carried out by 10 to 20 teams of 4 to 5 students each.

Iteration 1							Iteration 2				
Wk0	Wk1	Wk2	Wk3	Wk4	Wk5	Wk6	Wk7	Wk8	Wk9	Wk10	Wk11
startup	JRP1	JRP1	JAD1	JAD1/CC1	CC1	CC1	JRP2	JRP2/JAD2	JAD2/CC2	CC2	CC2

Figure 1: Project RAD organization.

In Week 0 (the startup week in figure 1), the teacher assigns students to teams. Team setup is made to homogeneously mix students according to their bachelor studies and their weekly availability. When the team first meets, it decides on the leader and the roles and tasks assigned to each team member. The students usually identify two role types: the *programmer* writing the code and studying the technology, and the *documenter* writing the requirements specification and the design, and the documentation in general (e.g. javadoc, test reports and installation notes). Each student might cover one or both types of roles, depending on the specific team organization.

The project teams compete for a prize (the SE contest) that is awarded for the best software application. At the end of the project, the winner is selected based on usability and originality criteria. Besides eternal fame, a small prize is offered by an industrial sponsor, handed over in a public ceremony.

2.2 Culture

Dutch students like being independent. They value a certain degree of freedom in the way they manage work and the assignment of roles and tasks. The same holds for the project assignment. For example, in 2005 the students had to develop an application helping foreigners to understand or learn about the Netherlands. This could have been a.o. a game, a teaching application, or an exploration system. The contents were left free as well. Some teams developed games to learn geography or history. Others developed a navigation system to travel around the nation. Freedom in the project increases motivation.

At the same time, Dutch students are very good managers. They are rarely late for lectures, they come to project meetings on time, they take their responsibilities. This quite often results in good teamwork.

Our students traditionally do not have very high technical skills. Generally, the students majoring in CS have the highest programming skills and the willingness to take on the programming roles. However, this does not guarantee sufficient interest in programming. Together with the fact that software engineering is often erroneously equated with programming, this has a negative side effect. The know-how about the application is completely left to the programmers in the team. Instead of assuming an active role deciding on requirements and design alternatives, the documenters in the team play a reactive role

and merely document what the programmers decide to program. This results in less work for the documenters and more work for the programmers. In addition, the learning effect is lower in that "everybody does what she/he was already good in".

Lastly, Dutch society does not generally reward high grades. This results in a low level of competition. Students often expect a large amount of work ensures a sufficient grade, even if the resulting deliverables are of very poor quality. The teams really compete for the SE contest, though, for the trophy, and the public image of being the project winner.

3 Lessons learned from editions 2003 and 2004

From the 2003 and 2004 editions we draw the following conclusions:

Multidisciplinary student population. Students from different bachelor programs clearly have different interests and know-how. We can classify the five types of students as follows:

- **CS.** Students majoring in *CS* are relatively technical, typically prefer programming tasks and want to concentrate on technical aspects of the SE process.
- **BMI.** Students in *Business Mathematics and Informatics* also are somewhat technical, but have business interests as well, and want to develop more managerial and organizational skills.
- **AI.** Students in *Artificial Intelligence* are somewhat technical as well. In addition, AI students are more interested in human and social aspects of computer science.
- **BI.** Students attending *Business Informatics* have little interest in technology, they rarely select programming tasks, and prefer management or marketing tasks. They often take on the team leader role. They are very good presenters.
- **MMC.** Students attending *Multimedia and Culture* have little to no interest in technology or programming. They often carry out documentation tasks. They have interest in the more social aspects of the SE process or the software application under development by studying human-computer interaction topics.

The above classification results from the observations we made during the two project editions. We can group the different types of students as follows (see figure 2): the *technical students* having SE skills and the interest to learn more in-the-field (CS), the *non-technical students* with few SE skills and more interest to learn managerial, organizational and social aspects of the software industry (BI, MMC), and an intermediate, *other* category that exhibits both aspects (BMI, AI). In some of the analyses below, we use a slightly different classification, in that we then distinguish students with a *technical* orientation (CS), students with a *business* orientation (BMI and BI), and students with a *social* orientation (AI and MMC).

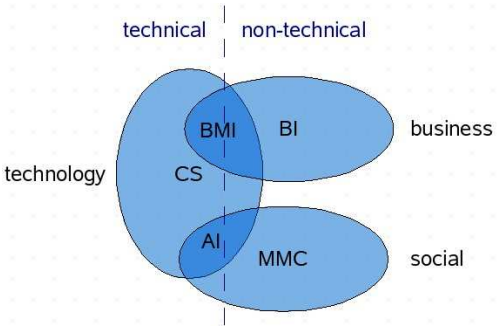


Figure 2: Student population and their classification.

By having a multi-disciplinary population it is difficult to develop a project that is exciting and interesting for everyone. In 2003 and 2004 the project assignments where technical, and concerned the development of an application with a given list of requirements. Business, organizational and human factors were not included. This made them less interesting for the non-technical students, to the detriment of their involvement, appreciation and learning.

Fear for technology. Our institute is well-organized and provides good support to its students. In the long term, a drawback hereof is that our students expect everything to be fully explained. Self-study becomes more difficult for them, and is loathed from the very beginning. In the PSE, for example, the students must autonomously study software environments, tools or downloadable Java libraries they never used before. This is perceived as a big obstacle, and is always subject to criticism. As a consequence, non-technical students prefer not to program, partially because they think they are not skilled enough.

Lack of SE maturity. In addition to the well known problem of putting SE theory into practice, our PSE students have little SE maturity. They did program in-the-small, but have no experience in e.g. reusing COTS or translating an ambiguous problem definition into a list of software requirements. Courses scheduled earlier in the curriculum formulate their assignments in a way that the software solution can be directly recognized. There is no need for brainstorming, for requirements elicitation, for decision making. The PSE is a big shock to them. For the first time the assignment is ambiguous, multiple alternative solutions are possible, and they have to decide themselves what to do, why, and how to do it.

4 Requirements Description

After the 2004 edition, we analyzed the observations listed in section 3, to improve the project for the next edition. The main question we decided to address, was: "How can we involve non-technical students more?". With the past experience in mind, we identified the following new requirements:

(r1) Increase the interest of the students with a more appealing assignment.
(r2) Take advantage of the MMC students' skills of usability and user interaction design.
(r3) Take advantage of the MMC technical knowledge of multimedia interface design in VRML for 3D animations.
(r4) Take advantage of the positive cultural characteristics of the student population, that is their independence and their abilities in teamwork management.
(r5) Take advantage of the BI students' skills in giving effective presentations, creating an impact on people.

As described in the next section, requirements (r1) to (r3) have been considered in the design of the new assignment, whereas requirements (r4) and (r5) led to modifications in the project organization.

5 Design

The design of the 2005 project edition is summarized in figure 3. It aims to attract students with diverse interests. Fragments of the 2005 assignment are described below to explain the corresponding design decisions and the requirements they realize. The 2006 project edition is similar to the 2005 one, in both the RAD organization and the type of assignment.

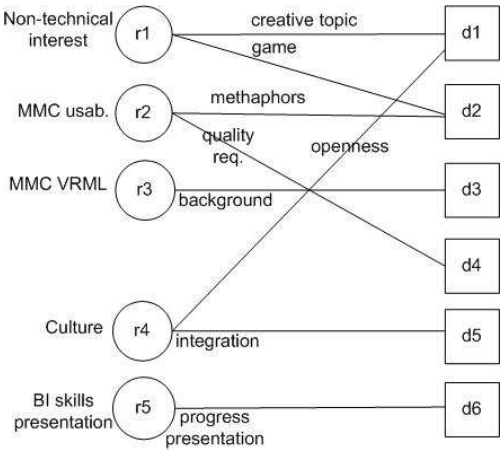


Figure 3: Mapping requirements and design decisions.

Creative topic. *"The assignment is to develop a software system that helps a foreigner to understand or to learn something about the Netherlands."*

This is a non-technical assignment, demanding creativity and teamwork where ideas are brainstormed and jointly evaluated. The assignment is also very open so that both the type of application and its contents can be autonomously proposed. This assignment fragment realizes requirements (r1) and partially (r4) (see figure 3).

(d2) Fun factor. *"The system must be presented as a toy: it should be funny, and use a metaphor (for example, use famous Dutch persons, objects or characters; be concrete or abstract) to teach in a funny and nice way the topic chosen."*

To develop a game increases the motivation of the average student. The objective of creating something entertaining and funny is a challenge that all students like. This assignment fragment mainly realizes requirement (r1). Further, it mentions the use of metaphors, a concept known to MMC students. This addresses requirement (r2).

(d3) Programming language. *"Technical constraints: Java (any further API's or Java-based technology is allowed, provided that it is free and downloadable)."* and *"[...] The system can make use of the Java VRML/EAI."*

This assignment fragment realizes requirement (r3), attracting the non-technical students with the possibility to put into practice their knowledge of VRML.

(d4) Qualities. *"Quality requirements are: usability and integrability."*

The MMC students have knowledge about how to elicit and model usability requirements. They know how to interview users, and how to specify their mental model to drive the design of a GUI. Integrability prepares the teams for the second iteration in which pairs of teams have to merge their applications into one (see design decision (d5)). This assignment fragment mainly realizes requirement (r2).

(d5) Application integration. *"In the second RAD iteration, pairs of groups have to integrate their respective systems. The integration must result in a software application which looks like 'one' system."*

Criteria like topic, GUI technology, system architecture, are used to pair the teams so that their applications are compatible. This fragment realizes requirement (r4) in two ways. Firstly, the "integrated" teams are relatively large (8 to 10 members)

and represent a management challenge to the team leaders and a collaboration and communication challenge to all team members. Secondly, each team decides on how to integrate their two software applications into a single system. After the first iteration, in which they develop a system from scratch, they now have to decide what to reuse, why and how to integrate. This again challenges independence and teamwork.

(d6) Progress presentations. *"During the project each team must give a progress presentation. [...] The presentations are intended to be interactive. Questions and/ or comments are welcome."*

In the previous project editions, each team had to give two presentations. The contents and structure were standard. The new progress presentations exploit better the presentation skills of the "business" students (as classified in figure 2). Progress presentations are scheduled when the iterations are still on-going: hence, the presenting teams can still include constructive comments, and share/solve open problems with other teams. Further, business students are better skilled in addressing an audience and communicating a certain message. To exploit this, we replaced the standard presentation structure with open content: the teams are free in selecting and motivating e.g. the most critical or innovative aspects of their results. This fragment realizes requirement (r5).

6 Implementation

At the end of each project iteration we summarized our observations and lessons learned. These are mainly based on what we could directly observe during the project, on the feedback given by the student assistants, and on the personal evaluations that each student writes as one of the final project deliverables. The observations for editions 2003 and 2004 are given in section 3. In edition 2005 the students wrote free-text personal evaluations, from which we got a very positive feedback. In edition 2006 we designed a questionnaire explicitly addressing a number of hypotheses, discussed below.

6.1 Quantitative observations from the 2006 edition

The assignment in 2006 was to develop an application supporting the user in learning and understanding different cultures in a global world. The winner of the SE trophy developed a game to travel around the world and play a context-sensitive quiz about different cultures in four continents. The team consisted of two CS students, five BMI students and two BI students.

In the 2006 project edition, we aimed at quantitatively testing the following hypotheses for the two types of students, i.e. technical (CS) and non-technical (BI, MMC) students:

(H1) Level of involvement. Technical students are more involved in SE projects than non-technical students.
(H2) Level of learning. Technical students learn more from SE projects than non-technical students.
(H3) Level of appreciation. Technical students appreciate SE projects more than non-technical students.

	Project task	
Study	*technical*	*non-technical*
AI	1	0
MMC	1	0
BI	2	5
BMI	3	12
CS	12	2
other	0	0

Table 1: Task versus specialization

To test these hypotheses, the students filled in a questionnaire about their type and perceived level of involvement, perceived level of learning, appreciation of the project as a whole, appreciation of their team, and personal technical proficiency. Further, we used the grades of the students. We received 38 filled-in forms back, from a total of 45 students that finished the 2006 project course. All questions used a 6-point Likert scale (1 = low, 6 = high).

When we apply ANOVA analyses for the differences between levels of involvement, learning and appreciation between the groups of technical and non-technical students, we obtain p-values between 0.14 and 0.39. We therefore reject the hypotheses that students with a technical study have higher levels of involvement, learn more or have higher appreciation of the assignment.

However when we discriminate between students with a technological, a social, and a business background, we do observe small differences between levels of involvement ($p = 0.068$) and the amount of learning ($p = 0.077$). Having observed overall differences, we compared the groups pair-wise using Tukey's honest significant difference procedure with confidence level 90%. In both cases the only significant difference was between the groups of students with business and social backgrounds. Students with a business background consistently outperformed

students with a social background when it comes to involvement, learning and appreciation. Apparently, the revised project still appeals less to students with a social background.

Further, we investigated if there was a relation between the study type (technical versus non-technical) and the type of project tasks the students chose. The type of tasks versus study specialization is depicted in table 1. There is a significant relation between study and the task chosen in the PSE ($\chi^2 = 15.83$, $df = 4$, p-value $= 0.003$). Technical students choose technical tasks, non-technical students choose non-technical tasks. This confirms what we observed in the 2005 edition.

The results regarding the level of involvement, the level of learning and the level of appreciation that have been presented in the previous section should be considered as results of explorative data analysis. The small sample size (n=38), combined with unbalanced groups in the ANOVA analyses, lead to results that are less than fully reliable.

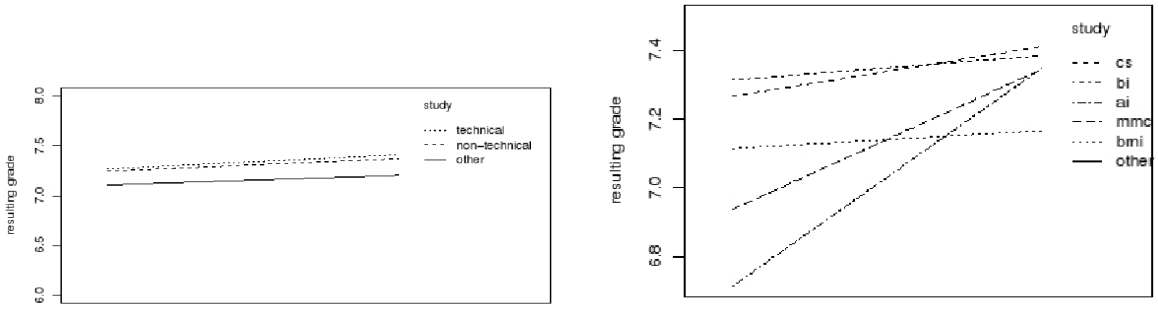
6.2 Performance improvements

In the years 2003 and 2004, the assignments were very technical. In 2005 and 2006, the assignment was less technical, to cope with some of the observations we made. With this change in focus, we hoped to improve the performance of the non-technical students. We therefore investigate the following hypothesis:

(H4) Level of performance. Non-technical students perform better on SE projects that have less technical assignments (2005-2006) than on those with more technical assignments (2003-2004).

To be able to compare grades, we first test to see whether the average grades are constant over time. An ANOVA-test indicates that such is not the case ($p = 0.0573$). So average grades do seem to depend on the year in which the software project was completed. When we however compare the grades of the technical assignments (i.e. 2003 and 2004 taken together) with those of the non-technical assignments (2005 plus 2006), we see no overall significant differences ($p = 0.339$). Because of that, we are able to use the latter combination of grades for a deeper analysis.

To analyze (H4) further, we investigate whether performance differences can be observed for all types of students, or just for some of them. We classify students as either technical (CS), non-technical (MMC and BI) and other (BMI and AI). The result is depicted in figure 4. The level of performance does depend on study type ($p = 0.284$), but the type of assignment (technical versus non-technical) does not seem to have a significant effect. All three study types perform somewhat better for the non-technical assignments.



(a) (b)

Figure 4: a) Average grades per study type, b) Average grades per study specialization.

We next look at the grades for the individual specializations of students: CS, MMC, BI, BMI, and AI. The result is depicted in figure 4. From this, we learn that the "social" types of students (AI and MMC) improve much more than the others do. So the social, cognitive elements from the improvements have had an impact, whereas the business/organizational aspects have had a minor effect, if at all. So requirements (r2) and (r4) in particular have been realized. Requirement (r3) has not been realized, in that VRML was considered too difficult, and was therefore not used. Overall, the non-technical project still appeals less to the social types of students. Rejection of (H3) (technical students appreciate SE projects more) supports

achievement of requirement (r1): apparently, we managed to make the project equally interesting to the various types of students. We may corroborate the same for (r5).

7 Related work

From the last years' papers published in the field of SE Education and Training we notice a shift towards the non-technical aspects of the SE process. The aim is at educating the old "laboratory nerds" with new skills in communication and problem solving. The special issue on educating software professionals [8] mainly addresses the theory-versus-practice approach. Van Vliet [16] favors the reconciliation of the engineering dimension with the human and social dimensions in SE. Lethbridge [11] made a survey on the knowledge important to SE professionals, and the participants declared that in their education they learned least about software management, business, and people skills.

A number of papers report about how effective these more social aspects can be taught in SE project-oriented courses.

Navarro et al [12] experimented with a project made up of one very large team (32 students) with an open assignment. Brereton et al [2] investigated the factors influencing project outcomes. As variables they used team size, gender, the abilities within the team, and the range of non-technical skills. Results suggest that weak students do not perform better if grouped with more able students, and that team size, gender and non-technical skills have no influence on outcomes. In particular, there seems to be no correlation between the study specialization and the outcomes. This is slightly different from our results, according to which a less technical assignment helps social-type students in performing better.

Robillard [13] reported about his experience in teaching a course similar to ours, and gave seven recommendations for a successful project-oriented course. Much emphasis is put on teaching teamwork including personal, technical and managerial issues. We share part of his experience, like: motivation increases if the team does good planning and reasonable workload distribution; students often have difficulties in admitting on-time that they have a problem. We form the teams consisting of heterogenous student types from the very start. In this way, all teams potentially expose the same overall skills. This on average lowers the chance of weak teams.

Verkano et al [18] focused on distributed cross-cultural SE. They carried out a case study in two universities in Finland and Russia. Surprisingly enough the students encountered no problems due to differences in either background or work culture: the initial difference in the students' attitude towards team hierarchy and documentation was naturally solved during the project.

A number of universities ([14], [7]) are renewing their SE curricula to include less technical subjects. Some address the social and communication aspects of SE per se.

There are few articles describing the design of modules addressing specific pedagogic goals. Most of them take the point of view of either technological support ([3]) or technical contents. A still low but increasing number of publications especially focus on the cultural implications associated with SE education in a global environment, like [6, 9, 1, 10].

In this paper, instead, we focus on the diversity in the student population. Along the same lines, van der Hoek et al [15] present the Informatics curriculum recently revisited at their university. This is organized around four SE education principles, namely: place software in the context of the information it manages; treat SE as a design discipline; place technical solutions in the context of the social structures where they will operate; and place the SE education's focus on synthesis in the context of analysis. They also address various study types, from CS to Information Systems, SE and Computer Engineering. In doing so, they observe that the concerns of SE students are more on development and technology than the concerns of e.g. IS students that are on deployment and organization & system issues. They suggest that modern SE curricula should educate not just programmers but also e.g. designers without their having to be master coders. This suggests that we should put more emphasis on project management and design, and treat them as disciplines equally important to programming and technology.

8 Conclusion

In this paper we discuss how we designed a multi-disciplinary software engineering project-oriented course. After the 2003 and 2004 editions we identified a list of requirements aimed at attracting and involving less technical students. We applied a number of changes to the next two editions, in 2005 and 2006. We then investigate if the changes improved the level of involvement, learning, appreciation, and performance of our technical and less technical students.

The 2006 data analysis suggests that the perceived level of involvement, learning and appreciation does not differ between technical and non-technical students. Using a different classification, we do observe a difference between students with a business orientation and students with a social orientation: the former consistently outperform the latter. Apparently, the revised project still appeals less to students with a social orientation (AI and MMC).

If we look at the grades, however, we observe that our changes towards multi-disciplinarity had a higher effect on the social-type students. We also found an overall slight improvement in all types of students. So the social, cognitive elements from the improvements have had an impact, whereas the business/organizational aspects have had a minor effect, if at all. Notwithstanding this, the social-type students still have a lower perception of the project course, when it comes to level of involvement, learning, and appreciation.

Further, there is a natural distribution of tasks that best fit the personal skills of students. This is confirmed by the quantitative data collected in the 2006 edition. This in our opinion increases the students motivation, appreciation and performance. We cannot say if this helps or hinders the level of learning.

Software engineering projects pose many challenges, both technical and non-technical. Conversely, a multi-disciplinary student population has its challenges too, caused by the different backgrounds and interests of the students. In this paper we show how the careful design of a software engineering project allows us to address both aspects simultaneously. Our focus has been mainly on involving less technical students in to software development issues. In the same vein one should pay attention to how to involve more technical students to social issues, too.

References

[1] B. Al-Ani, A. Hadwin, and D. Damian. Instructional design and assessment strategies for teaching global software development: a framework. In *ICSE*, volume 28, pages 685–690, Los Alamitos, CA, USA, 2006. IEEE Computer Society.

[2] P. Brereton and S. Lees. An investigation of factors affecting student group project outcomes. *Proc. of the Conference on Software Engineering Education and Training*, pages 163–170, 2005.

[3] J. Cabot, F. Durán, N. Moreno, A. Vallecillo, and J. R. Romero. From programming to modeling: our experience with a distributed software engineering course. In *ICSE*, pages 749–758, 2008.

[4] R. Dawson. Twenty Dirty Tricks to Train Software Engineers. In *ICSE*, pages 209–218. IEEE Computer Society, 2000.

[5] R. Dawson, R. Newsham, and B. Fernley. Bringing the 'real world' of software engineering to university undergraduate courses. *IEE Proceedings – Software*, 144(5-6):287–290, 1997.

[6] O. Gotel, V. Kulkarni, C. Scharff, and L. Neak. Working across borders: Overcoming culturally-based technology challenges in student global software development. *cseet*, 0:33–40, 2008.

[7] N. Habra. Peopleware integration in an evolving software engineering curriculum. *Proc. of the Conference on Software Engineering Education and Training*, pages 102–110, 1997.

[8] T. B. Hilburn and W. S. Humphrey. The impending changes in software education. *IEEE Software*, 19(5):22–24, 2002.

[9] P. Kruchten, Y. Hsieh, E. MacGregor, D. Moitra, W. Strigel, and C. Ebert. Global software development for the practitioner. In *ICSE*, pages 1032–1033, New York, NY, USA, 2006. ACM.

[10] P. Lago, H. Muccini, L. Beus-Dukic, I. Crnkovic, and S. Punnekkat. Gseem: Teaching software engineering in a european and global setting. *International Journal of Engineering Education*, 24(4):19 pages, 2008. Special Issue on Trends in Software Engineering Education.

[11] T. Lethbridge. What knowledge is important to a software professional? *IEEE Computer*, 33(5):44–50, 2000.

[12] E. O. Navarro and A. van der Hoek. Scaling up: How thirty-two students collaborated and succeeded in developing a prototype software design environment. *Proc. of the Conference on Software Engineering Education and Training*, pages 155–162, 2005.

[13] P. N. Robillard. Teaching software engineering through a project-oriented course. *Proc. of the Conference on Software Engineering Education*, pages 85–94, 1996.

[14] M. Shaw, J. Herbsleb, I. Ozkaya, and D. Root. Deciding What to Design: Closing a Gap in Software Engineering Education. In P. Inverardi and M. Jazayeri, editors, *Software Engineering Education in the Modern Age*, pages 28–58. Springer Verlag, 2006.

[15] A. van der Hoek, D. Kay, and D. J. Richardson. Informatics: A novel, contextualized approach to software engineering education. In P. Inverardi and M. Jazayeri, editors, *Software Engineering Education in the Modern Age*, pages 147–165. Springer Verlag, 2006.

[16] H. van Vliet. Reflections on software engineering education. *IEEE Software*, 23(3):55–61, 2006.

[17] H. van Vliet. *Software Engineering - principles and practice*. John Wiley & Sons, Ltd., third edition, 2008.

[18] A. I. Verkamo, J. Taina, T. Tuohiniemi, Y. Bogoyavlenskiy, and D. Korzun. Distributed cross-cultural student software project: A case study. *Proc. of the Conference on Software Engineering Education and Training*, pages 207–214, 2005.