Tandem Courses – Students coaching students

Görel Hedin, Lars Bendix, Boris Magnusson, and Lennart Ohlsson, *Computer Science*

***Abstract*—Teamwork and project participation are essential skills for professional software engineers. To be able to give large groups of students adequate opportunities to learn such skills at a reasonable teaching cost, we have developed two courses that work in tandem: a team programming course taken by about 100 students, and a coaching course taken by about 20 students. In this paper, we describe our view of how these courses should be integrated within the engineering curriculum and our experiences of running them for five years. Important aspects of our set up include co-location during development, fixed working hours, and guided reflection exercises for both coaches and developers. The projects are run as role-playing games where teachers acting as customers provide opportunities for negotiation and the making of commitments. Our experience so far is very positive, and we see that students get a good basic understanding of the important concepts in software engineering, rooted in their own practical experience.**

I. INTRODUCTION

THE development of software products is a complex task, involving many interrelated activities like requirements analysis, design, implementation, testing, documentation, release, and deployment. However, in software development it is well known that specifying the system to be developed so that it accurately reflects the client's true requirements is at least highly difficult and often even impossible. Established ways to overcome such obstacles typically involve prototype development, partial deliveries and tight loops of feedback from the users.

To teach these skills and concepts it is not sufficient to use the standard academic teaching model because the students need practical experience in order to get a deep understanding of the activities. It is therefore common to use project courses as the educational medium for these topics, typically involving students working together in teams [6]. However, in order to thoroughly learn these skills, the teams need a lot of educational resources in terms of guidance and coaching, something which is always a limiting resource in academic education.

We have therefore taken the opportunity to also offer a course on coaching, given in tandem with the basic project course. In the coaching course older students serve as coaches and project leaders for the younger students, and in this course the focus is on leadership where the students learn theory about leadership and teamwork, and get to practice it in a realistic setting.

Manuscript received April 18, 2006.

All authors are with Department of Computer Science, Box 118, S-221 00 Lund, Sweden (e-mail: {gorel|bendix|boris|lennart}@ cs.lth.se).

II. SETUP OF THE TANDEM COURSES

The tandem courses are given in parallel during two study periods of 7 weeks each, and the students take other courses in parallel as well. The Team Programming course is a pass/fail mandatory course of 4 credits (6 ECTS) for 2nd year students, and is taken by around 100 students each year. The Coaching course is a pass/fail optional course of 6 credits (9 ECTS) for 3rd-4th year students, and is taken by around 20 students each year. The students are studying for a Masters of Science in Computer Science and Engineering, and they are used to quite dense and intense courses.

We have run these tandem courses now for five years in a row. To get started, we ran the first instance of the Coaching course as a graduate course with a few additional handpicked senior undergraduate students. The next year we ran the Coaching course as a regular course with the Team Programming course as a prerequisite.

A. Theory and project parts

As the software engineering basis for the courses we use the methodology of eXtreme Programming (XP) [2], which is one of the most well known examples of the class of modern so called agile methodologies [1] and is centred around 12 practices.

The first study period is a *theory part* where the two courses are taught separately. In the Team Programming course, the theory of XP is covered and the students also have some lab exercises on tools and techniques to be used later in the project. In the Coaching course, teamwork and leadership theory is studied, both general team theory, and specific theory related to software development and agile work processes. Weekly homework assignments are given where the students reflect on the theory.

The second study period is a *project part* where the two courses are joined and around 10 programming teams are formed. Each team consists of 8-10 developers from the team programming course, 2 coaches from the coaching course, and 1 faculty member serving as a customer to the team.

The coaches also do an in-depth study of their own choice that is started during the theory part and completed during the project part. Often, they make use of the teams to gather experimental data or for trying out specific ideas.

B. Project iterations

The project is run as a series of 6 weekly iterations. An iteration starts on a Wednesday with a two-hour planning meeting where the team follows up on the previous iteration and plans for the next one. The following Monday, the team meets in one of the labs for 8 hours of program development (a “long lab session”). The customers (faculty) visit each

team both during the planning meeting and the long labs for discussion and prioritization of requirements.

Between the planning meeting and the long lab, each developer is expected to spend around 4 hours of "spike time", preparing for the long lab, doing experiments and studying specific issues. Exactly what each developer should do during the spike time is decided by the team at the planning meeting. However, actual program development takes place only during the long lab sessions when all members are present.

The coaches meet with faculty each Tuesday to discuss experiences from the previous iteration and to get advice for the upcoming iteration.

C. Expected results and grading

All teams develop essentially the same product and are given the same set of user stories (customer requirements). However, they are not expected to complete all the stories. The focus is rather on maintaining high quality in their work and on learning by doing.

In the last (7th) week of the study period, the resulting products are demonstrated and evaluated. This is done by peer evaluation, letting each team try out another team's product and review their code, technical documentation and user manual. These reviews are presented orally in groups of three teams at a time.

How can we grade a course that is time boxed and builds so much on teamwork? We grade it only pass/fail, and for passing they are required to have actively participated in all the scheduled activities. Students can apply for an exemption from the scheduled activities due to illness or exceptional circumstances. In order to make up for such absence, the students get additional tasks that are valuable for their team and which are presented to both the team and us.

More details about the two courses are available in the following papers: [3, 4, 5].

III. Experience

Software development requires very many different skills and at many different levels. In order to not overwhelm the students we think that an iterative learning environment is absolutely essential. It is necessary for the students to get the chance to do concrete work according to their current understanding and to get the possibility to reflect and get feedback from others in order to improve their understanding and their skills. This has turned out to be a very illustrative example of Kolb's learning cycle [7] for experiental learning.

The use of XP provides excellent opportunities for learning in this way. From a pedagogical perspective we identify four important aspects that in particular support the learning cycle: *time-boxing*, *iterations*, *co-location*, and *role play*. We now comment on each of these in turn.

A. Time-boxing

In most other course projects we give fixed requirements that the implemented system should fulfil. The Team Programming course is very different. Here we give the students a fixed time box (planning time, spike time, and long lab time), and expect them to do their best with this time. Completing stories is important, but not at the expense of quality of work, team communication, and peer learning. We find this approach absolutely essential in order to make sure that the students learn and practice various skills rather than focus on the product they produce.

B. Iterations and guided reflections

At the start of the project, the students have read about XP in theory, and although they have some practical experience from the labs, they have on the whole a passive and superficial understanding of the techniques. The course design with six project iterations gives them many chances to experience and reflect on how the practices work, to get feedback from their coaches and peers, and to improve during the next iteration(s).

Although the 12 XP practices seem fairly simple in principle, they take time to really understand and master. In order to not get overwhelmed by trying to learn everything at once, we have introduced *focus practices*. The idea is that while we instruct the students to try their best at following all the XP practices, for each iteration we select 4 practices that we ask them to focus on in particular during that iteration. They are asked to refresh the theory for these practices before they go to the long lab session, and at the subsequent planning meeting they are asked to discuss and reflect on their experience from these practices, and to suggest ways of improving. We ask the coaches to summarize these discussions, and each week we compile a summary and post it on the web for everyone to share.

C. Co-location and peer learning

We have found the practice of having the whole team co-located in one room extremely important for learning. We have heard of other experiments in applying XP in education where the teams are just left to themselves to find a common time and place for programming. This usually results in the team splitting into fixed pairs, splitting the stories between them and programming them at different times without much communication, learning, or reflection.

A co-located team, on the other hand, allows all team members to keep up with the development of all parts of the product, and promotes team building. Short "time-outs" are taken to discuss and reflect over both design and methodology with the whole team. Switching pair partners can be done frequently, supporting peer learning.

The teams are put together in a random fashion. Usually, the students initially say that they would prefer to form teams with their friends, but after the project, they agree that they probably learned a lot more by being part of a random team. Many students comment that the use of random teams makes the project feel more realistic.

D. Role play

An important element in project courses is to use the opportunity for role playing games [6]. The benefit of this old idea, which is based on Winnicott's thoughts on play and creativity [8], is that it provides a safety harness which allows you to test new and different behaviour habits without risking a blow to your ego. Giving students clear roles therefore allows them to learn more than if they were

required to fulfil a certain task. We use role play on several levels. In the XP practice of Pair Programming the partners have distinct roles: driver and navigator, which ensures continuous communication concerning all aspects of programming, from design ideas to mundane bookkeeping tasks. This communication promotes peer learning at all levels during programming. Frequent switching of the roles within the pair ensures that both partners get to experience being the "driver".

The coaches also take on different roles during the project: e.g., coach and project leader. Initially, they also have the roles of architect and tracker, but these roles are gradually delegated to the developers. The coaches can also go into the developer role to pair program with the ordinary developers, a very effective way of sharing their expertise, both concerning the methodology and concerning programming as such.

During planning meetings, the customer participates in the role play and the plan for the next iteration is negotiated: the developers estimate stories, and the customer sets the priorities and adds new stories.

E. Course costs

The two courses have many scheduled hours for the students. However, because of the use of student coaches, the needed faculty resources are reasonable. The two tables below summarize the number of teaching hours used.

Normal preparation hours are needed for the lectures and laboratory supervision, whereas the customer role requires much less preparation. Additional teaching costs include a one-hour exam, administration of the course, administering "focus practices", and handling absence, i.e., keeping track of students that are ill, come in late, etc., and taking appropriate actions. The costs below do not include the creation of stories for the product since we have reused approximately the same set over the 5 instances the course has been given.

Costs for Team Programming course:

Type of personnel (role)	**Description**	**In-class hours for *t* teams**	**In-class hours for 10 teams (100 students)**
Senior faculty (lectures)	7*2h lectures + 1*2h concluding lecture	16	16
Teaching assistants (lab supervisors)	4*2h labs (serving 2 teams simultaneously)	4**t*	40
Teaching assistants (customer in project)	6*2h customer at planning session (serving 2 teams simultaneously) + 6*4h at development sessions (serving 4 teams simultaneously)	12**t*	120

Costs for Coaching course:

Type of personnel	**Description**
Senior faculty	Lectures: 7*2h
Senior faculty	Supervision of coach meetings: 7*2h
Senior faculty	In-depth studies: Feedback on preliminary abstracts + Feedback on preliminary versions + 4h final seminar.

IV. Conclusion

Software development is a complex area, and success depends on many different skills, including programming skills, organizational skills, and people skills. Many of these skills require extensive practice in order to get a good understanding of the whole picture and the complex interrelations between different tasks.

We find the use of tandem courses allows us to provide the students of the Team Programming course with a very good learning environment, while at the same time providing the Coaching course students with the opportunity to learn and practice coaching under realistic conditions.

While both XP and our courses focus on software development, we think there are many aspects that would carry over to other areas where products are built or designed. Coaching and leadership skills are highly useful for engineers, and the tandem course setup should be applicable also to project courses in other fields.

Acknowledgment

We would like to thank the participants of our courses for valuable feedback and discussions concerning the course contents and setup.

References

[1] *The agile manifesto*, http://agilemanifesto.org.

[2] K. Beck, *Extreme Programming Explained – Embrace Change*, Addison-Wesley, 1999.

[3] G. Hedin, L. Bendix, and B. Magnusson, *Introducing Software Engineering by means of Extreme Programming*, in Proceedings of the International Conference on Software Engineering, Portland, Oregon, May 3-10, 2003.

[4] G. Hedin, L. Bendix, and B. Magnusson, *Coaching Coaches*, in proceedings of the 4th International Conference on eXtreme Programming and Agile Process in Software Engineering, Genova, Italy, May 25-29, 2003.

[5] G. Hedin, L. Bendix, and B. Magnusson, *Teaching eXtreme Programming to Large Groups of Students*, Journal of Systems and Software, January 2005.

[6] C. Johansson, L. Ohlsson, *An Attempt to Teach Professionalism in Engineering Education*, World Conference on Engineering Education, Portsmouth, 1992.

[7] D. A. Kolb, *Experiental Learning: Experience as the Source of Learning*, Prentice-Hall, 1984.

[8] D. W. Winnicott, *Playing and Reality*, Penguin Books, 1985.