Introducing Software Engineering by means of Extreme Programming

Görel Hedin, Lars Bendix, Boris Magnusson
Department of Computer Science
Lund Institute of Technology
Sweden
{gorel|bendix|boris}@cs.lth.se

Abstract

This paper reports on experience from teaching basic software engineering concepts by using Extreme Programming in a second year undergraduate course taken by 107 students. We describe how this course fits into a wider programme on software engineering and technology and report our experience from running and improving the course. Particularly important aspects of our set-up includes team coaching (by older students) and "team-in-one-room". 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.

1. Introduction

Teaching software engineering is difficult since many of the practices are motivated by large projects and large organisations of which the students have little or no experience. As many other universities we have previously been running an introductory software engineering course based on a small project. The project has been run in a scaled down waterfall style in order to illustrate basic concepts such as requirements analysis, specification, design, implementation, testing, documentation, and delivery. This traditional set up, with little or no support from faculty during the project, has always been problematic. In particular, it has been difficult to motivate and enforce all the practices for such a small project. With only one iteration, the waterfall model has no place for students to improve. The result has too often been that a few students on each team have been doing most of the work, drowning in details and deadlines rather than grasping concepts, and the rest of the team has learned even less.

In spring 2002 we changed the course completely, renaming the course to Programming in Teams (PT) and adopting Extreme Programming (XP) as the basis for the course. Although our gut feeling is that all of the XP practices are very valuable, our main goal was not to teach XP per se, but to use XP as a vehicle for teaching the basic concepts in software engineering. In running a practical project there is the need for following a method, and in using XP as the basis we have found it easy to give the students a solid basic understanding of not only the important concepts mentioned above, but also of, e.g., iterative development, configuration management, and team communication. In particular, the highly iterative nature of XP, where the participants get rapid feedback at all levels, makes it ideal in a teaching situation.

As a side effect, the PT-course also gives the students practical skills in testing, configuration management, and refactoring, they can immediately use in their projects in later courses. We find such skills to be much easier to teach in the new XP-based set up of the course.

The PT course is designed to fit into the larger picture of the curriculum where many different aspects of software engineering and technology are treated. We view the curriculum as being composed of three layers. At the bottom layer there are first and second year courses teaching basic programming and object-oriented design. In these courses the students work individually or in pairs. At the middle layer we find the PT course, which is focused on programming in small teams. At the top layer we find courses that address software development issues in larger organisations and/or more advanced topics.

The content of the classical introductory software engineering course is placed in the top layer, while the students only have experience from individual work thus creating a mismatch that is hard to bridge. By having the practical experience from the XP-based PT course, we think that the students later on can easier appreciate, understand, and critically discuss the concepts taught at the third level. At the top layer we also find a Coaching course – the second course in our scheme. Here the students learn about team coaching and software architecture. Each student in the coaching course acts as a coach for one of the teams in the PT course. The top layer also includes advanced software engineering courses on, e.g., requirements elicitation, configuration management, etc. In all of these courses the students can benefit from their

practical experiences from a small, yet complete, project in the PT course.

The rest of this paper is structured as follows: In section 2 we give a short background on the XP methodology and explain how we have adapted it for the PT course. In sections 3 and 4 we give more in-depth treatment of our experiences from applying the different practices in the course. Section 5 elaborates on the lessons learned and compares with related work. Section 6 concludes the paper and gives directions for future work.

2. The Team Programming Course

Course format

The course is organised in a theory part followed by a project part. This fit nicely into two consecutive study periods of seven weeks each (corresponding to the spring term).

The first part is a theory part with 6 lectures of 2 hours each covering: introduction; overview of XP practices; testing and pair programming; configuration management; simple design, refactoring, and architecture; and planning and estimation. At the seventh lecture the students were given a one-hour quiz that they must pass in order to be allowed into the project part of the course. Also, the project and product to develop were introduced at this lecture.

During the theory part there are also 3 lab sessions of 2 hours each in order to introduce the main practices and the tools. These labs covered: planning (which was conducted as a so called Extreme Hour [1]); test first and pair programming (using JUnit); and configuration management (using CVS). During these labs the students work in groups of 15-20 students, presence is compulsory and a short quiz is done at the beginning of each lab in order to ensure that the students are sufficiently prepared to get full benefit from the lab. The Extreme Hour was done in a classroom, whereas the other two labs were done in the computer lab working in pairs.

The second part of the course is a project part where the students are grouped into teams of 8-10 students that do an XP-style project in 6 iterations during 6 weeks of study. Presence is compulsory for all group activities during the project. Each team was coached by a pair of older students, who were, at the same time, taking the Coaching course. In the last (7^{th}) 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 documentation. These reviews were presented orally in groups of three teams at a time.

An iteration starts on Wednesdays with a two-hour planning meeting where the coach and the team follow up on the previous iteration and plan for the next one. The following Monday, the whole team and the coach meet in the lab for 8 hours, a "long lab session,„ of program development. In the meantime, each student is expected to spend around 6 hours of "spike time,„ on experimenting and studying particular issues. Spikes could be done individually, in pairs, or in larger groups. However, actual program development (which is checked into the common team repository) takes place *only* during the long lab sessions. Thus, each student spends a total of (2+6+8)*6=96 hours on the project, or 2.4 man weeks spread out over the 7 week study period, not counting the final evaluation.

As main literature we chose the book "XP Installed,„ [2], and supplemented this with some articles [3, 4, 5, 6]. We find the book a good practical introduction to XP, but it is very different from the normal text books that students usually have. It is aimed at practitioners and explains a particular method (XP) without any broader outlook or objectivity. We solved this by presenting the book as a "handbook,„ for the particular method used, rather than the objective final truth.

In the first course instance there were 12 teams of a total of 107 students. Students were assigned to labs and teams in a random fashion, with the goal of making everyone's schedule work out. All teams developed essentially the same product (a system for tracking races in the motorcycle sport Enduro) and were given the same set of requirements (user stories in XP). All teams also had the same person as customer (one of the authors).

In most other course projects we give fixed requirements that the system should fulfil. This 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 the best of this time, producing as much as they can of the requirements (user stories), while learning as much as they can in the process.

How can we grade a course that is time boxed and builds so much on team work? We grade it only pass / no pass, and for passing it is required to be actively participating in all the scheduled activities. Students could apply for an exemption from the scheduled activities due to illness or exceptional circumstances. In order to make up for their absence, these students got additional spike time tasks that were valuable for their team and which was presented to both us and the team.

XP as used in the real world

XP is a so-called agile methodology [11] promoting a highly iterative work model with the aim of producing high-quality software and allowing quick adaptation to changing requirements. It is centred around the 12 practices of: Planning Game, On-site Customer, Pair Programming, Collective Code Ownership, Continuous Integration, Small Releases, Metaphor, Simple Design, Coding Standard, Testing, Refactoring, and Forty-hour Week [7].

In a typical XP project the team may consist of around 6-8 developers, a coach, and a customer that is "on-site,„ and therefore available for informal discussions. The development work will typically be organised in short itera-

tions of around 2-3 weeks between which planning meetings are held. The intervals between releases is also short, for some projects on a daily basis and for others after a few iterations, the goal being to let the customer give feedback as often as possible and to be able to steer the project direction.

XP in an educational setting

The PT course is set up to fit into the normal study periods at our university. The study year is divided into 4 periods of 7 weeks each, and each student normally follows around three courses at the same time. The students are studying for a Master of Science in Computer Engineering, and they are used to quite dense and intensive courses.

In our educational setting there are a couple of major differences from a real XP project. First of all, we have accelerated the iterations substantially. Instead of letting the students do normal 2-3 week iterations, they do six 1-day iterations of development (during a 6 week period), and 3 releases (one release at the end of every second iteration). This way, they get several rounds of feedback through the repeated iterations and releases, while not having to spend an unreasonable amount of time on development.

Secondly, the students are in a learning situation. Not only do most of them have no experience from being part of a software development team, but they also have quite limited programming experience in general. Therefore, they need help both with what to do regarding the project, but also with programming as such.

There are several other differences as well such as the developers being graded (pass / no pass) rather than paid, the customer being a professor rather than a real customer.

Additional practices

Our goal has been to apply the 12 XP core practices more or less according to the book. In addition, we have applied a number of practices, most of which are also described by XP practitioners, but usually not mentioned among the core practices. These are: First Iteration, On-Site Coach, Team-in-one-room, Spike Time, Reflection, and Documentation. Because we run a number of iterations we had the opportunity to tune and refine the practices somewhat during the project.

3. Experience from core XP practices

In this section we will detail the experience we have from the core XP practices. The available space does not allow us to go into details with all 12 core practices, so we have selected some that have interesting or surprising lessons.

Pair Programming Three of the student coaches performed a thorough study on the PT students' attitudes and experience from pair programming during the project [8]. They conclude with some advice on how to use pair programming in an educational situation, e.g.:

- The coach can recommend developers to pair in a certain way in order to spread competence and work efficiently. However, it is the developers themselves that ultimately should decide on who to pair with.
- Personality matching seems to play a more important role than competence matching for a pair to work well.
- The coach should strive for assuring that everyone on the team pairs with as many of the others in the team as possible. This is important from a learning perspective and team building. However, certain pairs might not work well socially, and in such cases they can pair less often and take on simpler tasks in order to not cause bottlenecks for the project.
- Sometimes the two developers in a pair come into a conflict on how the code should be developed. In this case, the coach can step in and resolve the conflict, or call on a stand up meeting for letting the team as a whole resolve the conflict.

Some groups established a pattern of almost fixed pairs, while others changed partner more often. One group even made it a virtue to mandate a change of partner after each task. There were major differences in programming experience, but in most cases that did not seem to cause problems. Sometimes the coach would have to split a couple that were in crisis with respect to solving a task to get in "new blood„ – or to simply reassign the task to another pair.

Planning Game From the beginning many students were somewhat sceptical at estimating tasks without any prior experience. However, they took on the challenge and gained practical experience on how to estimate new tasks. Most of the groups continued to do estimates and planning right to the end, but some groups did, however, not bother to do estimates towards the end – "we'll just do tasks and see how far we get„.

Some of the teams estimated in hours and others in relative size of stories. Of the 40 hours available (5 pairs for 8 hours) during a long lab session, one group consistently got 25-30 estimated hours of work done. Especially the first iteration served to show the students that things took longer than they had thought.

A total of 35 stories were created for all of the 6 iterations, and most teams completed around 28-30 stories. New stories were created before each iteration and on some occasions the customer changed his mind on priorities between stories, or changed and extended data formats. This was not planned beforehand, but we found it a nice realistic illustration of how customer requirements do change.

Test First Some groups did test first by the book and some groups almost did it. However, in other cases unit tests were written after the code had been written. In general the students had big problems in understanding how to write unit tests in a useful way. Often some small changes to the code would also require that some unit tests needed to be changed or rewritten because they were too specifically tied to the implementation. At other times there would be unit tests that worked at too high a level and therefore did not catch more subtle bugs. For the next instance of the course we will try to remedy this by giving more training on writing good unit tests.

Small Releases The students did three releases – one every two iterations. The first release was done to the customer and included installation and user manuals. The customer was overwhelmed by work in evaluating and commenting the 12 releases, so for the subsequent releases they had to release to a peer group to relieve the customer. An additional advantage of this was that the students were forced to see the release from the customer's point of view. For these releases they should also include source code and documentation, not just the executable and manuals. In many groups the first release went wrong, so they did a new "first„ release after iteration three as well (plan to throw one away).

On-site Customer Since the students are fairly inexperienced programmers and in a learning situation, we wanted to have good control over the user stories. There should be a sufficient number of stories so that several pairs could work in parallel, and be reasonably simple to implement so that a pair of students had a chance of getting through each story in one iteration (one day of programming). To accomplish this we needed a customer with XP experience. We therefore settled on not bringing in any outside customer, instead one of us acted as the customer. We picked a product where the customer had deep domain knowledge due to his leisure interests (the Enduro MC sport) to make the product and requirements realistic. It was a realistic option that the products developed by the students could in fact be used for a real Enduro competition.

Since we had only one customer, all teams had to share him. The customer made short visits to each team during both the long lab sessions and planning meetings. However, for spontaneous questions and discussions he was less available, and the coach therefore often acted as a stand-in customer.

4. Experience from the added practices

In this section we describe how we have applied the added practices and our experiences from them in a teaching perspective.

First Iteration The First Iteration practice [9] says that in the beginning of a project it is very important to build up an executable skeleton of the system, forming an initial thin but complete system. The idea is to make sure early-on that the necessary infrastructure can be built and that subsequent development can proceed in small chunks. This idea is somewhat similar to the Elaboration phase in the Unified Process, at least according to the recent interpretations of it as of Craig Larman's "Agile UP„ [10]. In XP, First Iteration serves the role of getting early feedback on the system architecture. If some architectural aspect of the system is not working, it is better to evolve the architecture early on before too much time has been invested in building functionality. For this reason, First Iteration is also called "Zero Feature Iteration„.

Building a First Iteration requires experienced programmers and would have been very difficult for our novice students to do. We therefore applied First Iteration by letting the coaches build a tiny first iteration of the system. The PT students could then immediately start adding functionality to the existing system. To aid the coaches in this initial phase we suggested three possible architectures of increasing complexity and suggested that they should start with the simplest one and later evolve it to the more advanced ones, if needed.

On-Site Coach The importance of having a coach, in particular when starting out with XP the first time, has been stressed by many XP practitioners [7, 1, 9]. In a situation of learning or adopting a new set of practices it is important to do a serious job of actually applying the practices. It is all too easy to forget or simply dismiss a practice while concentrating on the actual development. The coach knows the practices, can explain them and motivate why they are important, and can also help the team to actually enforce the practices. Once the practices are mastered the individual is in the position of challenging them and adapting them. We think this is an important way of adopting a new methodology regardless of whether it happens to be XP or something else.

In the accelerated process that we are working with in the project it is important that the coach is always available and actively follows the activities of the team members. The coach needs special skills and training and we therefore came up with the idea of creating a course for third and fourth year students on coaching software teams. The coaching activity is the trainee part of the Coaching course that also covers theoretical aspects of these areas. A pair of students from this course serve as coaches, project leader, tracker, and chief architect for a team of students doing the XP project. Before each iteration we have a 2 hour meeting with the coaches to discuss their experience from the previous iteration and give instructions for the next one.

During the long lab sessions, the coach constantly keeps track of what the team members are doing, "stresses„ them when that is needed, and "caresses„ them when that is appropriate. An important goal for the coach is team building, to make sure that the team members communicate and collaborate for the common product.

The first time we gave the PT course we naturally had no students with XP experience to enrol on the Coaching course. In order to "bootstrap„ the process, around half of the coaches were teachers and Ph.D. students and the rest were hand picked undergraduate students that we believed would be interested as well as do a good job. The student coaches more than fulfilled our expectations, not the least considering team building. In this first instance of the coaching course, we included some condensed XP training with literature studies and discussions as well as lab sessions. There is a lot to be said about the coaching role and the coaching course, but that will be the focus of a forthcoming paper.

Team-in-one-room Real XP projects are often set up so that all of the members share an open workspace where program development takes place [3]. We adopted this practice: program development takes place only at the long labs where all members of the team are present in the same lab room. We found this to be an extremely important practice.

First of all, the students can easily communicate so that everyone is aware of what is happening, and help from fellow students is always near at hand. For example, when one pair got stuck they could immediately get help from others on the team, switch tasks, switch pairs, etc., in order to more easily solve the problem at hand.

Secondly, being all in one room allowed the coach to easily spot any problems, be it a specific programming task, some pair not following the practices, or a situation when the coach could see that two pairs needed to communicate. On such occasions the coach would take a "time-out„ i.e., gather the team for a short stand-up meeting to discuss the problem and find a solution. Rather than telling the developers exactly what to do, the coach would on such occasions ask the developers to come up with a solution. This way, the coach helps the developers to learn to help each other, and soon the developers would themselves call on stand-up meetings when needed.

Thirdly, being all in one room very strongly promotes team building. The teams were put together randomly so initially the students did not know each other well. However, due to all the interaction during the long labs, most of the teams soon developed a very strong social team spirit which again promoted further communication and awareness.

It may not always be easy to schedule full day labs for students who take several classes at the same time. However, since these are 2nd year students, most of their schedule is still fixed by mandatory courses, and it was therefore possible for us to negotiate a schedule that did not interfere with other courses. For scheduling reasons, one third of the teams did their long lab from 1-10 PM (rather than from 8AM-5PM). Scheduling was a bit of a problem for some of the student coaches, but this was handled by letting them coach in pairs, so that at least one coach was always present for each team.

Spike time In addition to program development, the developers need time to do individual work, e.g., to read and learn about various practical issues, to think about how to solve a particular task, or to do experimental programming with new solutions. We termed this "spike time„. During each planning meeting, the team allocated this spike time in the way they thought best for the team as a whole, thereby providing a pool of additional resources. We gave the coaches some initial ideas for how to allocate the spike time, and after some iterations the following patterns emerged:

- Spiking for a task. In order to be able to program a particular task at the long lab, the students experiment with some key aspects, for example find out what Java libraries to use, and write or find some simple example code that can be used as a starting point later in the long lab.
- Reading/learning about tools. In some teams, a "CVS expert„ was appointed whose job it was to use all his/her spike time for learning about CVS in order to help the team mates at the long lab. In other teams, all team members were allocated a couple of hours for experimenting with CVS. Similar spiking was done for other basic tools like JUnit and makefiles, and also for tools introduced during the project such as a refactoring tool for one team, and an acceptance testing tool for another team.
- Refactoring. After a couple of iterations many teams found the need for doing large refactorings of the code. This was difficult to do during the long labs since it affected and/or delayed many teams and produced many merging conflicts. Several teams adopted the policy of allocating spike time for such refactorings. In these cases, refactorings were usually done by a single pair.
- Code review and unit tests. Some teams found that they had too few tests, and used spike time for doing systematic review in order to find code with missing unit tests and to remedy that.

For each team an additional CVS repository was set up that included results from the spikes. This way, the developers could easily share their results with their team members. Not only code was checked in, but also short manuals and other material that the developers produced during their spike time. The general rule was to not check in to the regular repository during spike time, since changes there should only happen when all the team was present at the long labs. However, many teams developed the practice that certain tasks that were allocated to spike time, e.g., big refactorings and adding test code, could be checked after agreeing before-hand at the planning meeting.

The coach meetings (when all coaches met and discussed the previous and next iteration) allowed the coaches to share ideas for how to use the spike time and resulted in many new ideas that we had not been able to foresee. At the last iterations, there was less need for

spike time because development had become very intense, and the developers needed more time for development rather than for spikes for new tools or new tasks. However, there was always the need for increased code quality, which motivated e.g. spike time for writing more test cases and refactoring.

Reflection The Agile Manifesto [11] lists reflection as one of the principles in order to follow an agile methodology: *"At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behaviour accordingly"*. In our XP project, there were several occasions when such reflection occurred spontaneously, e.g. at stand up meetings called on by the coach in order to discuss a particular practice, and at the planning meetings when the results of the previous iteration were discussed. On such occasions it was often decided to do a spike to avoid that a problem repeated itself.

In the next instance of the course we will introduce reflection as an explicit practice so that all teams make a habit of doing some reflection over their work practices at each planning meeting. It is our experience that the spontaneous reflection that occurred in some teams, greatly helped in team building, as well as improved the effectiveness of the team.

Documentation XP does not prescribe any specific documentation, but views the code as the most important (and always up-to-date) documentation. However, core XP can of course be extended with additional practices to cater for appropriate additional documentation. In our XP project, we let the customer write a user story demanding a technical documentation of the system, including an architectural description, build description, description over internal file formats, etc. This was very easy to motivate since the customer might like to take the finished source code and let another team develop it further at a future time.

Several teams also spontaneously developed additional internal documentation like instructions for how to do a release, story test descriptions, etc. Part of this documentation was developed during spike time.

5. The lessons

We have learned many things from running the PT course in this new format, and we summarise our most important lessons below.

Team-In-One-Room is fundamental

Of all the additional practices we introduced, the practice of having the whole team in one room was perhaps the most important of all. 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, in which case it 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. In these cases they usually also get problems with collective code ownership and resort to code owning practices [12]. We believe Team-In-One-Room is a fundamental practice to apply in an educational setting, and that if it is not possible to schedule full day lab sessions like we did, splitting into a couple of half day lab sessions is better than splitting the team.

Coaching by older students

In novice teams there is a need for someone with a bit more experience who can take an overall perspective on both the product architecture and the team practices. We had very good experience from applying older students as coaches. This way the coaches had some authority both due to their general seniority and in that they were given special training in the coaching course, yet they were not too distant socially from the PT students which made it easy for them to build a good team spirit. The coaches also found it very stimulating to take on the needed responsibilities.

Underestimated difficulties for the students

During the course it became apparent to us that we had underestimated the difficulty of several practices.

- The "Test First„ practice was much more difficult to explain and teach than we initially thought. There is a need for very clear examples on how to write tests, and we will try to include that in the next instance of the course.
- The "Simple Design„ practice was also more difficult to teach than we expected. Some students misinterpreted "Do the simplest thing that can possibly work„ to mean "Change as little as possible to incorporate a new feature„„ resulting in a bad design. Here, there is again need for many good examples (and maybe also anti-examples) and to follow up this better during the project.
- A related point is refactoring. Only one of the teams used a refactoring tool in the project, and then only on an experimental basis. This lack of tools had the effect that the students were very reluctant to do refactoring, which again had a bad effect on maintaining a simple design. For the next iteration of the course we hope to introduce a refactoring tool, e.g., by using the Eclipse environment [13], and to give them lab exercises so they gain some confidence and proficiency in using such tools. We expect this to have a major impact on the designs.
- Both the students and we underestimated their problems in putting together a release. Fortunately though, we had scheduled for three releases. Several students thought it would take around 15 minutes to produce a release, whereas in reality it

took up to half a day for a pair, and many of the first releases did not even work. After this experience many of the teams allocated a pair at the next iteration to put together a "release process„ i.e. to build a checklist for how to put together a release and what things to check in order to be assured that it would work. Once that was in place, the release process went very smoothly for most teams and they could produce a new release in a matter of minutes.

Although we underestimated many difficulties and the students ran into many problems during their project, this is not necessarily a bad thing. On the contrary, we found it very valuable for the students to run into these problems so they could better appreciate their solutions. Due to the many iterations they got experience from both having problems and solving them which provides a good basis for learning. The preceding theory part of the course together with the coaching during the project did, however, provide the students with a reasonable background and framework so they did not run into all problems at once, but could feel that they did progress with each iteration.

Being a customer is much work

We underestimated how much work it is to be a customer. We needed a good amount of stories that were not too difficult and that could be done in parallel by several pairs. Due to the very short 1-day iterations the stories needed to be much smaller than for a real XP project. To accomplish this we broke down the larger stories into smaller tasks, although this is something that normally the team would do itself, and we also gave the teams directions for the initial architecture and possible ways of evolving the architecture. For the next instance of the course we plan to simply use the same product and stories as we had this year. As we get more experience with the course we might get bolder and bring in new products on the fly, and put more responsibilities on the coach like breaking down stories to tasks and giving them more responsibilities for coming up with the architecture.

XP might have positive effects for minority students

The high interaction and collaboration within an XP team, in particular when guided by a trained coach, might have positive effects for students in minority groups. At the computing engineering program, only around 10% of the students are women. Most women on the course had very positive experiences from the course, and anecdotal evidence points towards them more easily being able to take on active roles as compared to earlier courses.

6. Related work

There seem to be several experiments on including some of the XP practices into the undergraduate education, e.g. [14, 15, 16], as well as experiments with a more complete set of XP practices. Muller and Tichy [17] performed a study of two small 4th year student XP teams, focusing primarily on evaluating the XP practices. They confirm our observation that coaching is very important. Noll [18] provides some observations from initial experience on applying XP to student projects, confirming our experience that XP is an excellent aid in learning, due to its highly iterative nature, allowing students to make mistakes and learn from them. Both these studies also confirm our experience that the Test First practice is difficult to learn. Our study differs by being much larger in scope, both in the number of students and teams, and in the number of additional practices applied. Our study also differs in being focused on XP as an aid in learning software engineering concepts rather than finding out if XP as such is efficient or not.

Williams and Upchurch [19] provide some guidance on the use of the XP practices in an educational context. Most of their advice is in agreement with our experience. However, their advice on not using collective code ownership for novice students is in contrast to our findings. We found collective code ownership to be very important and not problematic at all. All our teams used CVS and were taught to synchronise frequently. In combination with the practice of keeping the whole team in one room, this is probably the reason for our success with collective code ownership. Some teams did initially not synchronise frequently, but the merge problems they then got made them understand how important this is.

7. Conclusions

When our course in introductory Software Engineering needed to be revised we set out to try to teach basic principles applicable on the team level and to use XP as the method. Other secondary goals of the course is to get the students more experienced as programmers, to better understand the problems of working in a team with others, interacting with a customer/user and to expose them to some new programming tools. The course was taught in full scale with 107 students with a background in fundamentals of programming, but no training in Software Engineering. It is followed by courses giving a more traditional software engineering perspective.

In contrast to earlier versions of the course we decided to take a larger responsibility for the management of the project part of the course, actually teaching a method, enforcing and supporting it during the project. For us it was a very interesting alternative to use XP as the method which we judged would suit this situation and would be a vehicle to expose the students to some fundamental software engineering principles and practices.

The result was very positive: we feel that the students have a good understanding for fundamental problems and techniques in software development in mid size teams including testing, inspection, interaction with user/customer, requirements, release process. They also understand how to work in an iterative method and were exposed to the problems of changing requirements. During the project they learned the problems of being many developers changing the same source and some mechanisms to handle these: configuration management, merge, short check-out times, refactoring and separated tasks. The students gave a very positive feedback after the course - it was at times very intense for them, but they report that they learned a lot and were very satisfied with the course.

Our experience is also that XP worked very well for this setting, introducing practices and techniques that could be motivated directly by the situation the students were facing. Having a blank table to start with we set out to use as much of XP practices as we found motivated, expecting that they would support each other. We did, however, have to do some amendments and additions due to the particular teaching situation. The most important additions we feel are to provide support for the architecture, having to do with rather inexperienced students, and thus providing a first iteration from the start and putting more of the responsibility for the software architecture on the coaches. We also broke up the work in smaller tasks than customary in order to have the students perform many iterations.

The format of the course needed to be adjusted to the use of the XP method and in particular we managed to schedule one full day each week for software development. We thus had the students in the same location for 8 hours and could use pair programming with changing partners. Only during this time was it allowed to develop production code. The set-up also was very important in order to build a team and to get all the participants to actually know each other well enough. Another unusual set-up was to use students of a second SE course as coaches for the teams. This allowed a much more intensive coaching than would have been possible using ordinary faculty resources, and provided a realistic setting for the coaches to learn practical aspects of architecture, team building, and leadership.

Acknowledgements
The development of the PT course and the coaching course has been a large collaborative effort by many people at the department. In particular, we would like to thank Torbjorn Ekman and Christian Andersson who came up with the initial idea of full day lab sessions, and for being very helpful in the implementation of the course.

8. References

[1] K. Auer, R. Miller, *Extreme Programming Applied*, Addison-Wesley, 2002.
[2] R. Jeffries, A. Anderson, C. Hendrickson, *Extreme Programming Installed*, Addison-Wesley, 2001.
[3] K. Beck, *Embracing Change with eXtreme Programming*, IEEE Computer, October 1999.
[4] K. Beck, E. Gamma, *JUnit Cookbook*, http://junit.sourceforge.net/doc/cookbook/cookbook.htm.
[5] W.C. Wake, *The Test Code Cycle in XP: Part 1, Model*, http://www.xp123.com/xplor/xp0002.
[6] W.A. Babich, *Software Configuration Management – Coordination for Team Productivity, Chapter 1*, Addison-Wesley, 1986.
[7] K. Beck, *Extreme Programming Explained: Embrace Change*, Addison-Wesley, 2000.
[8] M. Nystrom, J. Rix, K. Wanhainen, *En studie om parprogrammering i praktiken* (in Swedish), http://www.cs.lth.se/Education/LTH/01.Dokt.XP/Djupstudier/NystromRixWanhainen.pdf.
[9] W.C. Wake, *Extreme Programming Explored*, Addison-Wesley, 2002.
[10] C. Larman, *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and the Unified Process* (2nd Edition), Prentice-Hall, 2001.
[11] *Manifesto for Agile Software Development*, http://agilemanifesto.org/.
[12] L. Bendix, G. Hedin, *Summary of the Subworkshop on Extreme Programming*, Nordic Journal of Computing, Vol. 9, No. 3, Fall 2002.
[13] *Eclipse Platform Technical Overview*, Object Technology International, Inc., http://www.eclipse.org/.
[14] O. L. Astrachan, R. C. Duvall, E. Wallingford: *Bringing Extreme Programming to the Classroom*, in Extreme Programming Perspectives, Addison-Wesley, 2003.
[15] M. Holcombe, M. Gheorghe, F. Macias: *Teaching XP for Real: Some Initial Observations and Plans*, in Extreme Programming Perspectives, Addison-Wesley, 2003.
[16] D. H. Johnson, J. Caristi: *Extreme Programming and the Software Design Course*, in Extreme Programming Perspectives, Addison-Wesley, 2003.
[17] M.M. Muller, W.F. Tichy, *Case Study: Extreme Programming in a University Environment*, in proceedings of ICSE 2001, Toronto, Canada, May 2001.
[18] J. Noll, *Some Observations of Extreme Programming for Student Projects*, position paper at the Workshop on Empirical Evaluation of Agile Processes, Chicago, Illinois, August 7, 2002.
[19] L. Williams, R. Upchurch, *Extreme programming for software engineering education?*, in proceedings of the 31st ASEE/IEEE Frontiers in Education Conference, Reno, Nevada, October 10-13, 2001.