Extreme Programming in Curriculum: Experiences from Academia and Industry

Matthias M. Müller[1], Johannes Link[2], Roland Sand[2], and Guido Malpohl[1]

[1] Universität Karlsruhe, Am Fasanengarten 5, 76131 Karlsruhe
[2] andrena objects ag, Albert-Nestler-Straße 9, 76131 Karlsruhe

Abstract. Since the rise of the light weight software processes, the paradigm on how software should be developed has started to shift. Agile methods strive to supersede the traditional software process with its exhausting requirements elicitation at the beginning of a software development project, at least for smaller or younger companies.
The software engineering group at the Universität Karlsruhe has accounted for this shift and extended their offer of lectures by an Extreme Programming lab course held in cooperation with andrena objects ag.

Key words: Extreme Programming, programming lab course, curriculum, experience report

1 Introduction

In the last thirty years, the inevitable necessity of exhausting requirements elicitation combined with an overall design at the beginning of a software development project has dominated software development. The exponential growing defect removal cost curve was the rationale behind this paradigm. However, since the rise of the light weight software processes, the paradigm on how software should be developed has started to shift, at least for smaller or younger companies. From this perspective, it is quite natural that students become aware of other possible software development processes which seek to account for a rapid changing business environment, and software companies discover the alternatives presented by light weight processes as well. In this changing environment, curriculum at universities should reflect the new possibilities for software development.
Since the summer semester 2000, the software engineering group at the Universität Karlsruhe has accounted for this change and extended their offer of lectures. While the new Extreme Programming (XP) lab course initially aimed at getting in touch with the new process, it is now an inherent part of the summer semester's lectures. The first experiences about this course were reported in [1]. In the last two years since the cooperation with andrena objects ag, the course settings have changed. The reason for change was the feedback from students and the views of andrena's XP professionals. The change with the most impact was the decision to issue a project week where the students had to work from Monday

to Friday from 9 am to 5 pm instead of the weekly four hours programming sessions scheduled previously. Consequently, the course was cut into halves. In the first part, introductory sessions presented the basic techniques of XP such as pair programming, test-driven development [2], refactoring, and the planning game. Along the way, students got to know supporting tools like JUnit and CVS. For the second part, the project week, students were divided into equal sized groups of six members each. Each group had to develop a WWW based movie theater seat booking system.
While the course settings changed, the students' preferences and problems concerning the XP practices kept the same. Pair programming was adopted from the very first session. This experience is consistent with others [3, 4]. But, dividing the work into small increments and implementing them step by step in a test-driven fashion was difficult. However, the problems diminished over time and at the end of the course, even test-driven development was seen as a comfortable way of development by most of the students. Wilson [3] observed the same effect in his course: after the first difficulties were overcome students appreciated test-driven development.
The next section describes the programming course in detail. Motivation of the software engineering group and andrena objects ag for this course is given in sections 3 and 4, respectively. Section 5 accounts for the experiences.

2 The Programming Course

The course is presented from three different perspectives: the content, the organization, and the project.

2.1 Content

XP Practices Introduction of the XP techniques alternated between lectures and programming sessions. Pair programming was introduced by andrena's XP professionals. After motivating the usage of pair programming, they exemplified pair programming with test-driven development by the development of an account class. During the presentation, the students got a first notion of how pair programming works and what development in small increments means. From the first session on, every programming task had to be solved in pairs with changing partners for every new task. The following sessions introduced test-driven development with JUnit, refactoring, CVS, the planning game, the basics of servlet programming, and HttpUnit.

Supplemental Techniques Throughout the course, students used Java together with the *Eclipse* development environment. *Tomcat* was used as web server. It was managed by the Sysdeo Tomcat plug-in for Eclipse [5]. The choice for Tomcat was natural as the plug-in provides *start*, *stop*, and *restart* buttons for ease of use from within Eclipse. In the same manner, *JUnit* formed the testing framework. The integration of JUnit into Eclipse alleviated the adoption of

test-driven development as the technical overhead was kept to a minimum. Version control was done with *CVS*. Committing local changes to the CVS server was synchronized by a CVS token. Each group could choose its token individually. Finally, Eclipse packaged with Tomcat, JUnit, and CVS did not make it necessary to leave the development environment during implementation.
Acceptance tests were written using the testing framework *FitNesse* [6]. The core of FitNesse is a wiki-server which has to be started in order to use the test framework. As everybody in the team had to work on the same set of acceptance tests, a separate acceptance test server was made available for every group to ensure consistency. Writing acceptance tests was allowed only at the server, though, every pair could execute them locally on their own computer. Using a dedicated server for the acceptance tests synchronized different pairs all wanting to enhance the tests. Although only one pair could work on the acceptance tests at a time the students did not complain on this bottleneck.

2.2 Organisation

In the summer semester, lectures at the university last for thirteen weeks. The introductory sessions were issued in the first and the last four weeks of this time frame. Figure 1 depicts an overview of the course schedule. The time from the

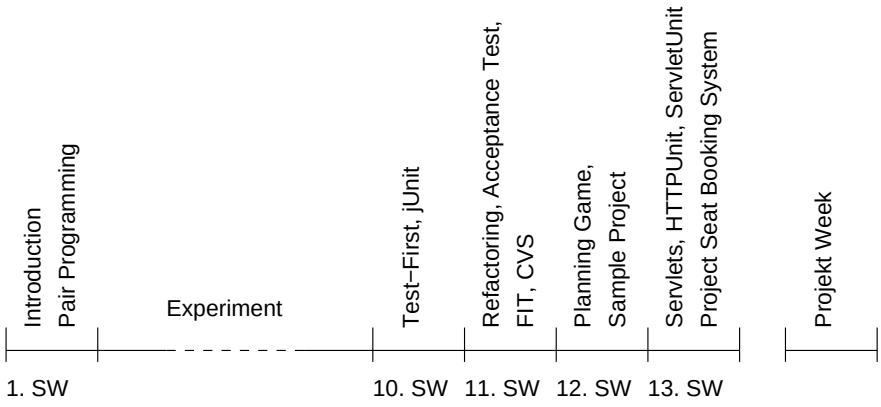

Fig. 1. Course schedule (SW = semester week)

second to the ninth semester week was reserved for a controlled experiment. Although the students had to participate in the experiment, the experiment did not add anything to the course topics.
The concluding project week was scheduled one week after the lectures. During the week, the students had to work from 9 am to 5 pm with about one hour lunch break. The project week was the largest change in the course settings

as compared to previous years where the course had a weekly 4 hour schedule. We established the project week because students startup time in the weekly meetings to familiarize themselves with the new techniques were large. As a consequence, students had to less time during the weekly meetings for working on the scheduled programming tasks. During the project week, startup times appear only at the beginning of the week and then, they cease to exist.
For the project, the students were divided into three groups of six members each. The groups were selected according to students own performance ratings which they had to assess in the subject questionnaire of the experiment. Each group had its own coach and customer. The roles of the coach and the customer were taken on by the course organizers. Each organizer coached one group and played the customer in another group in a round robin fashion.

2.3 Project

Each group had to implement the same project: a theaters online seat booking system. The system should provide a customer and a box office portal. The groups started with a small skeleton of the seat booking system which implemented basic operations like movie selection and ticket ordering. The skeleton was provided to alleviate the beginning of the project by giving the students a basic software architecture to start with. We decided to provide the skeleton due to our experience in the preceeding year when student teams started from scratch. Since the students had little or no experience with setting up a web project it took them almost a day to have the first visible results, i.e. a simple dynamic web page deployed on the tomcat web-server. This experience left the students frustrated with what they considered hardly an accomplishment after a full day of work.
In order to familiarize the teams with the provided skeleton, each group had to work on the following fixed set of stories during the first day.

Story 1 Logout for theater's box office.
Story 2 The number of tickets for one order is limited by five tickets.
Story 3 Announcement of new movies and more halls.

The stories had to be divided into smaller tasks. For the first iteration, division of stories was done by the organizers as well. The following list sketches additional tasks. These tasks had to be done only once at the beginning of the project.

Task 1 Configuration of work places such that all provided tests run.
Task 2 Configuration of acceptance test server.
Task 3 Reorganizing the code base into two Java packages.

Arrangement of work places is necessary for every project, though, time consuming. Thus, configuration of work places and test server has to be recorded. Task 1 and 2 account for this project set up. During Task 3, the group should separate implementation concerns of the customer and the box office which were mixed up in the initial code base and presented some kind of code smell.

Although the groups got the same skeleton to begin with and the course organizers agreed on a set of stories, the three projects evolved in different directions concerning functionality. The different characters of the three projects were not caused by XP itself but rather by the different priorities and notions the "customers" had on the stories.

3 Motivation of Institute

The software engineering group came across XP in the winter of 1999. Initial literature [7, 8] aroused our interest in the new process. Thus, we planned for a programming course to satisfy our curiosity. Our first aim was to get in touch with the process and to gather practical experience beyond the guidelines formulated in the literature. We focused on the promises made by XP. However, we were disenchanted with the process as there were lots of problems nowhere mentioned so far. The subsequent report [1] summarized our experience.
In the following time, we concentrated on the evaluation of XP. Focused on XP promises, we started to investigate the central implementation techniques. In order to achieve reasonable results, a better understanding of parts of the process and the process as a whole was necessary. We were triggered by the following questions:

- How can these techniques be modeled ?
- Where are the pitfalls while evaluating them ?

We started to educate students in XP and its techniques such that we could use them in experiments. Our results [9–12] show so far, that XP and its techniques are far from being the promised silver bullet, though, there are project settings where XP strengths come into full play and where XP is a reasonable choice.
Apart from the evaluation issues, the need for coaching experience grew because acquisition of background knowledge is a mandatory prerequisite for teaching compulsory lectures. And last but not least, presenting students alternative development techniques besides the conventional process is a welcome byproduct.

4 Motivation of Industry

andrena objects ag has been experimenting with XP since 1999 when the first online material became available on the Web. The main motivation for our interest was XP's focus on both quality and high flexibility. Some of the XP techniques were successfully adopted early on in in-house projects: test-driven development [2], pair programming, simple design, refactoring, continuous integration, short iterations and stand-up meetings. Other practices could not be realized as easily due to our customers' resistance, for example to provide a person who is full-time dedicated to support the development team. Moreover, we realized that familiarizing developers with test-driven design took lots of extra training and intensive supervision. Another difficulty we hit on was some developers' initial unwillingness to pair.

So our motivation to collaborate with the software engineering group was manifold:

- We wanted to gain experience in introducing XP to developers new to agile development practices. Sometimes we have to deal with beginners in Java or even programming who must be integrated into an XP team. Trying to teach students about XP and agile development seemed a reasonable way to practice teaching in a controlled and riskfree environment.
- In order to convince our customers' of doing XP we need enough empirical data and statistically relevant numbers to support XP's view of the development world. Helping researchers to gain insight into the subtleties and intricacies of XP is andrena's way to accelarate the appearance of relevant studies and publications.
- Since andrena objects focuses on high-quality individual software we are in need of developers who strive for a high standard in their development skills. Teaching at the University is one way to get to know future employees and to be visible for graduates looking for a job.
- Last but not least, giving lectures and courses about XP is a good means to "spread the word" and to make students acquainted with agile development.

Eventually, the management board of andrena objects decided to invest in student education by allocating two experienced developers to the XP programming course; so far, there have been no regrets.

5 Experience

Experience is shown from three different viewpoints: students' experience, software engineering group's experience, and andrena's experience.

5.1 Student's Experience

Pair programming was the technique the students' had the least problems to adopt to. This may be caused by the nature of the course itself because students subscribed voluntarily such that only those students took part who were willing to pair off for development. The "youth" of the students may be another reason for the fast adoption of pair programming. Most students are not coined by any software development paradigm. Thus, accepting new rules for implementation does not conflict with confirmed habits. And finally, pair programming causes a casual atmosphere on the sometimes boring development process.
When pair programming was the method adopted most easily, test-driven development caused the most problems. Breaking down the whole programming task into small steps was one problem. Implementing the small pieces in a test-first way was the other. At the beginning of the course the students asked for an idea on how to divide the problem. Later on, the answers switched over to on how this or that can be formulated in a test. But in the end, students felt comfortable with test-driven development. Work progresses step by step which reduces the

mental task stack to a minimum such that the possibility of forgetting something is minimized. As a byproduct, automated tests were seen as helpful and the number of new written test cases was seen as some kind of productivity measure.
The Planning Game was taught in the fourth introductory session, see Figure 1. In the mini project part of that session, students had to estimate and implement small tasks of about five minutes duration each. From this session, they came to know that effort estimation even for small tasks is difficult. As a result, in the beginning of the project week, students were uncertain about their story and task estimates. Another problem for story estimation was lack of overview over and experience with the code base. However, as the project proceeded estimates became more and more accurate because analogies to existing stories could be found.
Another aspect of XP is team software development. Most students do not have any experience in team software development. Thus, the experience that the own piece of code depends on the proper implementation of others and that the own code only runs if others' code runs as well is an experience most students are not familiar with. Other experience concern the team feeling. Students were proud of finishing an iteration with all the assigned stories done. It was seen as success of the team. And on the other side, no pair was blamed on not concluding its tasks, even though the customer was unsatisfied with the team's progress. The problem was identified later on and the whole team sought for solutions.
Every team was assigned its own acceptance test server. The customer formulated its acceptance test and the team had to implement them. However, writing acceptance tests with FitNesse is HTML centered. A small change to the HTML skeleton of the project web sites resulted in a dozens of acceptance tests that had to be adapted as well, although the functionality has been retained unchanged. But after two days, the students found a means to modularize the acceptance tests thereby minimizing the necessary rework from iteration to iteration.

5.2 Institute's Experience

From a programming course organizer's point of view, teaching XP is simple: follow the given rules. However, practice looks different. The subtle discrepancies between what is written in literature and how it is put into life are challenging. As a result, there is almost no field which we can identify as *the* field we got the most new experience in.
Besides these differences, the way XP handles unit testing is refreshing. Although students subscribed and participated in a programming course with a strong emphasis on XP as a development process, they got a keen sense of what can go wrong within the code. The emphasis on the automated tests widens the way students look on their own code. However, we do not believe that it is really necessary to implement the tests *first*. But for the classroom, test-driven development is the means for teaching software testing, as it forces students to think on how the implementation task can be divided into small and testable parts.

5.3 andrena object's Experience

The most valuable effect for andrena was that we – professional developers but no professional pedagogues – could refine our teaching skills. Especially we learned that the time it takes to bring an XP newbie up to speed can differ widely, but most students managed the basic techniques in the end. In contrast to the institute's view we are convinced that taking the test-first road has a big advantage. In our experience most developers write less and worse tests when adding them after the fact. Many even start "forgetting" them completely after a few iterations. This is mainly a psychological problem: writing tests for functionality that already works is perceived as useless and tedious whereas writing tests beforehand can be much more of a challenge.
Another point we eventually had to accept is that decent programming knowledge is a must; students who fight with Java syntax cannot focus on the development process at all. Luckily, the students' programming skills were better on average than what we can usually expect from "professional developers" in industry.
Effort estimation is difficult and must be practiced to reach a reasonable accuracy. As for standard development tasks the students quickly learned to come up with realistic numbers. However, as soon as something fundamental changed, for example a new technology was introduced, estimations were yet again very inaccurate.
Finally, some XP practices which we had somewhat neglected in our professional work are well worth the effort. Especially automated acceptance tests can give the development team much more confidence in what they are doing than fine-grained unit tests alone.

6 Conclusions

In the summer semester 2004, the Extreme Programming lab course will be offered for the fifth time. While the first course aimed at acquainting the software engineering group with the new process, the course is now an inherent part of the summer semester's lectures.
The changes in the course settings with the most impact are as follows.

- We switched from a course setting with a weekly 4 hour schedule to a course which is held en bloc to reduce students startup time needed to familiarize themselves with the new techniques.
- A skeleton was provided to alleviate the beginning of the project and to provide the students with a basic software architecture.

By teaching XP, we made the following experiences:

- Reasonable Java programming skills are a must. Students who struggle with Java syntax cannot focus on the development process at all.
- Pair programming is the technique students' have the least problems to adopt to.

– Test-driven development causes the most problems. Breaking down the whole programming task into small steps and implementing the small pieces in a test-first way is difficult. But in the end, students feel comfortable with it.
– Students feel uncertain during story estimation but over time, estimates become more and more accurate due to the ability of finding analogies to existing stories.
– Most students are unfamiliar with team software development. It is a new experience that the own piece of code can depend on the proper implementation of others people's code.
– Teaching XP requires practical experience. An experienced XP practitioner cannot be replaced by the study of literature.

References

1. Müller, M., Tichy, W.: Case study: Extreme programming in a university environment. In: International Conference on Software Engineering, Toronto, Canada (2001) 537–544
2. Link, J.: Unit Testing in Java - How Tests Drive the Code. Morgan Kaufmann (2003)
3. Wilson, D.: Teaching xp: A case study. In: XP Universe, Raleigh, NC, USA (2001)
4. Fenwick, J.: Adapting xp to an academic environment by phasing-in practices. In: XP/Agile Universe. Volume 2753 of Lecture Notes in Coputer Science., Springer (2003) 162–171
5. Sysdeo Formation: Sysdeo eclipse tomcat launcher plugin. (http://www.sysdeo.com/eclipse/tomcatPlugin.html)
6. Martin, R., Martin, M.: Fitnesse. http://www.fitnesse.org/ (2003)
7. Beck, K.: Extreme Programming Explained. Addison Wesley (1999)
8. Beck, K.: Embracing change with extreme programming. IEEE Computer (1999) 70–77
9. Müller, M., Padberg, F.: On the economic evaluation of XP projects. In: Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE), Helsinki, Finland (2003) 168–177
10. Padberg, F., Müller, M.: Analyzing the cost and benefit of pair programming. In: International Symposium on Software Metrics (Metrics), Sydney, Australia (2003) 166–177
11. Müller, M., Padberg, F.: About the return on investment of test-driven development. In: International Workshop on Economics-Driven Software Engineering Research (EDSER), Portland, Oregon, USA (2003)
12. Müller, M., Hagner, O.: Experiment about test-first programming. IEE Proceedings Software **149** (2002) 131–136