Best Practices in eXtreme Programming Course Design

Kai Stapel
FG Software Engineering
Leibniz Universität Hannover
Welfengarten 1
30167 Hannover, Germany
+49 - (0)511 - 762 7463
kai.stapel@inf.uni-hannover.de

Daniel Lübke
FG Software Engineering
Leibniz Universität Hannover
Welfengarten 1
30167 Hannover, Germany
+49 - (0)511 - 762 19672
daniel.luebke@inf.uni-hannover.de

Eric Knauss
FG Software Engineering
Leibniz Universität Hannover
Welfengarten 1
30167 Hannover, Germany
+49 - (0)511 - 762 19680
eric.knauss@inf.uni-hannover.de

ABSTRACT
Teaching (and therefore learning) eXtreme Programming (XP) in a university setting is difficult because of course time limitations and the soft nature of XP that requires first-hand experience in order to see and really learn the methods. For example, iterations are either shorter or fewer than appropriate. In this paper we present the properties to tune when designing an eXtreme Programming course. These are the properties we gathered by conducting three XP labs as part of our software engineering teaching. Within this paper we describe our set-up as well as the important properties. Lecturers and teachers can use this property system and combine it with their own constraints in order to derive a better XP lab for their curriculum.

Categories and Subject Descriptors
K.3.2 [**Computers and Education**]: Computer and Information Science Education – *curriculum, computer science education, information systems education*.

General Terms: Management, Design, Human Factors.

Keywords
Extreme Programming, Lab Design, Agile Development.

1. INTRODUCTION
Agile development, nowadays mostly represented by eXtreme Programming (XP), is being used more and more in industry. That is why at least one agile development practice should be in the repertoire of a university level student in a computing major.

However, it is hard, if not impossible, to realistically teach all relevant parts of XP in a University setting. This is mostly due to time and resource limitations. So the main differences between educational and real XP projects are that educational projects have either few but long or many but short iterations, simpler project objectives, and inexperienced programmers. Nevertheless, the goal is to get most out of an XP course for the student in terms of a realistic XP experience and XP learning outcomes.

ICSE'08, May 10–18, 2008, Leipzig, Germany.

In this paper we present how we tried to fulfill this goal and what we changed in course design to make things better. After putting this contribution in context with other educational XP papers we present a detailed description of our existing designs of XP laboratories. The experiences gained when conducting these XP labs are condensed in a property system which is being presented along with other best practices for eXtreme Programming course design.

2. RELATED WORK
There are many other educational papers concerned with XP course design and XP course experiences [1-5]. Most of the presented teaching and research is concerned with pair programming, as this is the most prominent XP practice. Probably that is because it is fairly easy to let students program in pairs instead of letting them program alone. According to Sherrell and Robertson [3], Williams and Kessler [4] and Hanks et al [5] students benefit from pair programming when learning programming, because they feel that they are more productive, their morale is higher and they are more confident in their programming solution. Besides that, students feel that pair programming makes them better at working with others [1].

Most commonly, classes, lectures and labs are concentrating on a selection of aspects, or even on a single aspect. For example, Astrachan et al [6] tried to create a lecture for XP, which concentrated solely on some practices like pair programming and refactoring, and Johnson and Caristi tried to integrate XP with software design [7] in their classes.

The main difference between our approach and most other approaches to teach eXtreme Programming is that we do not focus on a single aspect alone but instead we try to make it a holistic XP experience for the students. To achieve this we decided to make the course a block course (8 hours a day in consecutive work-days). The advantage of this is that learning of most of the XP practices (see section 3.1) is more effective when experiencing them non-stop. Schneider and Johnston [2] state that in most university settings it is not possible to get students to work co-located and continuously because of disjoint timetables and the lack of suitable laboratories. Luckily, we have an especially equipped computer room and timely-flexible students.

Our approach is influenced by Holcombe et al. [8] who also tried to set-up a more realistic project. In their case they had to teach XP to more students than we did, which led them to simulate a whole software company.

3. EXISTING DESIGNS OF OUR XP LABS

Within this section we present our approach for teaching XP in a lab environment. This includes the objectives we want to convey to the students, the lab set-ups we have tried and modified, and the required and resulting student qualifications.

3.1 Learning Objectives

The XP lab is one of the advanced classes that can be chosen by computer science master students. Beforehand, students had to participate in two programming classes (Scheme and Java), the lectures "Foundations of SE" and "Software Quality" as well as to perform a simulated software project that is based on a rigid software development process [9] in their bachelor studies. While XP and other agile methodologies are introduced in the lecture "Foundations of SE", the XP lab is a much more suitable form to introduce and teach XP and the feeling of agile methods.

Consequently, the main purpose of the XP laboratory is to let the students *experience* and conduct an agile development project. In that way they are able to *feel* the differences to process-driven development from a project participant's perspective. After taking the XP course, students should be able to judge which development style better suits a given development task, and whether they feel comfortable in participating in agile projects or not. Part of the XP experience is – besides the realistic XP environment – that students learn to perform and utilize XP practices [10] adequately. In contrast to lectures and other classes, our lab aims to convey the objectives practically. This is important because many properties of agile projects must be experienced in order to appreciate the advantages of doing so and to identify pitfalls that seem to be easy in theory [11]. Hence, the main learning objectives are to practically experience the following XP practices:

1. **Planning Game:** Students learn how to divide requirements into User Stories and how to prioritize and estimate the costs of these stories. While such tasks seem to be easy in theory, dealing with dependencies in the planning game is normally a challenge for inexperienced developers like students.
2. **Small releases:** Students learn the benefits of small releases that already offer value to the customer and how to technically put a system into production including packaging.
3. **Metaphor:** Students learn how to develop a metaphor that helps every team member to better understand how the whole system works.
4. **Simple design:** Students learn the benefits of simple software design which improves their ability to change the system quickly and accommodate it to changing requirements.
5. **Testing:** Students learn how to use unit test frameworks and the test-first approach to build high quality software and to recognize the advantages of well-tested code when making changes.
6. **Refactoring:** Students learn to refactor the software to remove duplication, improve communication and simplify the code base. Especially refactoring large systems can be troublesome and is a worthy experience that can only be made in long lasting projects.
7. **Pair programming:** Students learn and experience the principles of pair programming, the advantages of writing software with a partner and the social challenges that are associated.
8. **Collective code ownership:** Students get to know the advantages of collective code ownership and the challenges that arise with parallel updates and changes to their own code by other team members.
9. **Continuous integration:** To counter conflicting updates to the code base, students learn to integrate and build the software frequently.
10. **40-hour week:** In contrast to normal life in university, students experience to work continuously for 40-hours per week in a designated team room.
11. **On-Site Customer:** Students have to interact with a designated On-Site Customer who is available full-time to answer questions.
12. **Coding standards:** Students experience the importance of uniform coding conventions throughout the team especially when combined with collective code ownership.

Other important non-XP-specific learning objectives are:

- **General programming:** The students improve their general programming skills by practical experience.
- **Team work:** The students learn how to deal with differently skilled developers as well as social and technical problems faced by all teams

3.2 Different Lab Set-Ups

To realize the described learning objectives we started with a block course project that is integrated as a lab into the curriculum. Over the years we enhanced the set-up to improve the learning experience and to provide a realistic project experience. To achieve this, the student feedback was gathered by using a light-weight post-mortem technique called LIDs (light-weight documentation and reuse of experiences) [12]. In addition, we documented the experiences of the tutors responsible for the lab. This allowed us to improve the lab set-up over time.

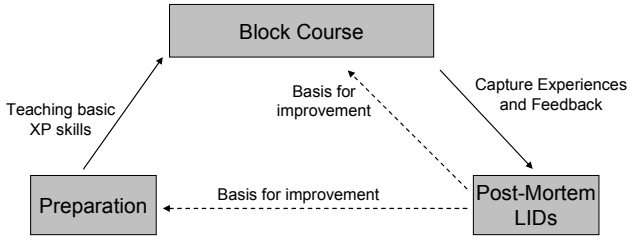

Figure 1: Feedback with LIDs for the XP Labs

Thereby, the basic project set-up remained the same as illustrated in figure 1. A preparation class conveys the principles of agile methods and XP practices to the students. Afterwards, a multi-day block course is conducted, which is the main part of the lab. The students have to develop an application for a customer using the XP practices. The On-Site Customer is played by a tutor. Another tutor performs the coach role. The experiences of the lab are documented

in a LIDs session afterwards. The experiences are used to further improve and adjust the next year's lab set-up.

Our first lab (in the year 2005) was a 3 day project that was accompanied by an introduction to XP. The introduction consisted of 2 lectures with 4 hours each. Besides a theoretical introduction into the XP practices and tips, agile hours [11] and practical introductions to the Eclipse IDE, refreshers for Swing and JUnit, using stubs for testing, and ANT were integrated into the lectures.

The main part was, however, the 3 day project. In this first attempt, each of the iterations lasted half a day resulting in 1 prototype phase and 7 iterations. The project took place in a designated computer room that was blocked exclusively for 3 days.

In this first attempt to conduct the XP lab, the XP team consisted of 16 developers, one On-Site Customer, and one XP coach. The coach was also responsible for answering technical questions regarding Java and other technologies. On-Site Customer and coach were performed by staff members of the software engineering group while the students were developers. Students had to work in pairs and the use of test-first was controlled by using test coverage metrics, which were published to the team.

Within the lunch break, the whole team including staff members ate pizza in a separate room designated to the team only. Moreover, after each day a small meeting was held in which every student had to offer his opinion on the day and his/her impressions. In this way we achieved a good team spirit.

While the main objectives of training XP practices and experience the "feeling" of an XP project were achieved, the initial set-up had some drawbacks.

- The iterations were very short leading to extremely fine-grained story cards that were too technical. This was both the impression of the attending tutors as well as the feedback given by the students.
- The overall project was very short. According to the students, the team did not work well before the third day. As a result, the team did not reach its productive state.
- The team size was too large. This led to many team members that were unwilling or unable to offer their opinions in discussions. There were too many students to effectively have common sessions at whiteboards. Especially, technical "group" discussions were always dominated by the same 3 students.

We tried to address these issues in the second lab that was conducted one year later. We extended the lab duration by doubling the associated credit points. Thereby we achieved a 7 day project with one day for the prototype and 6 iterations of one day each. We hoped to fix the problems with iteration length and the team building time this way. Due to personnel constraints we could not reduce the team size because we only had one On-Site Customer.

Interestingly, the second set-up worked worse than the first. The iteration length of one day seems to be an unsatisfying length. The story card size for such an iteration seems to be too large for being technical as with half day iterations but too short for being really customer-value driven. Moreover, that year's students were much more unwilling to communicate and to solve internal problems. Due to the increased number of days we could not eat pizza every day. Instead, most lunches were in the university's cafeteria which is more anonymous than a separate room. Thus, lunch breaks did not reach the team building effect as in the previous project. While in the first lab the lunch discussion was mainly dominated by spare time topics, the second group had nearly no talking at all. Although we expected positive effects of the increased iteration length and overall project duration the second attempt was worse than the first. This was also stressed by the fact that some students tried to avoid their accountability by hiding in the large team. Failed acceptance tests, deleted tests in the repository and non-working software were attributed to anonymous "other" team members.

A year later the XP lab was offered again. This time, we added another 2 days to the project length which was critical because 9 blocked days for a project means 2 weeks off from normal university classes for the students. By introducing a second On-Site Customer with a second project we were able to split the group and reduce the team size to eight students per team. In order to reduce the staff's workload both teams shared the same coach.

To improve the project and to better reach the learning objectives, we further increased the iteration length to two days while keeping all other things the same. In order to supervise and support two projects with one coach, the second project started one day later as illustrated in figure 2. Consequently, the coach supervised only one planning game each morning, because the teams alternatively started their iterations.

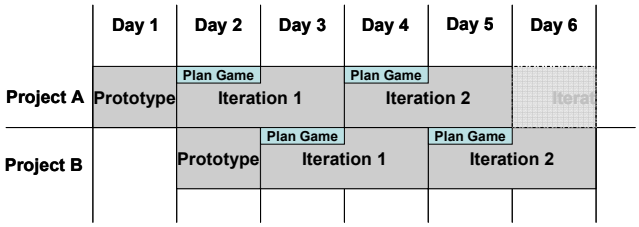

Figure 2: Interleaving Schedule for Two Concurrent Projects

On a side note, we organized the lunch differently: Only the first and last days of the projects were "pizza days" in the designated room. The other lunch breaks had to be organized by the students themselves. The organization requires communication between the students to select food and locations.

Using this set-up we have reached the best project and learning experiences in our opinion and feedback so far. The iterations are long enough to implement reasonable story cards and small enough to fit many iterations and therefore many planning games into the lab.

4. XP LAB DESIGN EXPERIENCES

Within this section we present our XP laboratory design property system. Based on our experiences, it helps lecturers planning to teach XP to align the lab set-up with the learning objectives. Additionally we give further hints and note worthy details at the end of this section. Thus, we conclude this section with recommendations and best practices.

Our experiences stem from observations we made during the various XP labs, the feedback of the students conducted in the LIDs after each lab, and some software metrics that were recorded on each check-in (tracking records).

Table 1: eXtreme Programming Lab Design Property System

Modifiers ⬀ better ⬂ worse ! Reinforce **bold** see text	**Learning objectives**															**Other effects**		
	Realistic XP environment	*1. Planning Game*	*2. Small releases*	*3. Metaphor*	*4. Simple design*	*5. Testing*	*6. Refactoring*	*7. Pair programming*	*8. Collective ownership*	*9. Continuous integration*	*10. 40-hour week*	*11. On-site customer*	*12. Coding standards*	*General programming*	*General team work*		*Student motivation*	*Tutor workload*
Block Course	**⬀**							⬀	⬀	⬀	**⬀**	⬀			⬀		⬀	⬂
Longer iterations	**⬀**	⬂						⬂		⬀								
More iterations		⬀	⬀		⬀	⬀	⬀			⬀								
Small team size (8-12)		⬀						**⬀**	⬀	⬀					**⬀**		⬀	
Customer interest	⬀	⬀					⬀					⬀						
Technical support		⬀		⬀	⬀	**⬀**	⬀	⬀						⬀			**⬀**	⬂
Technical feedback				⬀		**⬀**								⬀			**!**	⬂
Progress feedback		⬀	⬀														!	⬂
Lunch together															**⬀**		⬀	

4.1 XP Lab Design Property System

We performed an eXtreme Programming laboratory three times in the past three years. Each time we modified some constraints in order to improve the XP experience for the students. Some changes turned out to really have a positive impact on the learning objectives. Table 1 summarizes our experiences made when changing laboratory settings and the impacts of these changes on the learning objectives (see section 3.1). The results can be used as a framework for lecturers who are planning an eXtreme Programming course in a university-like setting.

The arrows in the table-cells represent the effect of the modification (row) on the learning objective (column). An upward arrow indicates a positive effect and a downward arrow indicates a negative effect. In other words: an upward arrow means that the modification makes the learning outcome more successful; vice versa for the downward arrow. For example, giving *technical support* helps the students to learn how to use a *metaphor* right, whereas *long iterations* usually are bad for the *pair programming* experience (since students tend to not switch pairs voluntarily during an iteration). The last two columns show the effect of the modifiers on two other non-XP lab setup relevant outcomes, namely student motivation and tutor workload. In the following paragraphs we give extended descriptions of some important modifications that we consider particularly important (marked bold in the table).

4.1.1 Block Course

We think that it is very important to teach agile development in a *block course*. Otherwise the project climate, the team building and the whole XP experience will not really be conveyed. Many of the XP learning objectives also benefit of a block course. Experiences regarding pair programming, collective code ownership, continuous integration, and On-Site Customer are more intense, i.e. the implications of those practices are better visible. Moreover, block course scheduling is the only option to demonstrate the importance of a 40-hour week. In a regular course the students would not get to feel how exhausting it can be to pair program 8 hours a day, 5 days a week. We assume that a block course would foster overall team work and typically would improve student motivation. This was also the feedback of the students. They think that the block course is a lab type that is well-suited for conveying XP. In our opinion the only downside is the very high tutor workload during a block course. Tutors working as coaches hardly get to do other things during the whole time of the course.

4.1.2 More and Longer Iterations

In a university setting it is almost impossible to have realistic iteration lengths (between 2 and 6 weeks) *and* at the same time a realistic number of iterations. So if you want to teach division of requirements into User Stories and cost estimation of these User Stories, you need at least a couple of iterations. That is why shorter iterations typically are used. We tried three different iteration lengths: half-day, one day, and two day iterations. The *longer the iterations* are, the more realistic the XP experience is. That is mostly due to the fact that User Stories should be of a size so that they can be completed in one iteration and add customer value to the system when finished. With short iterations User Stories need to be smaller so that they can be completed. Otherwise, the planning game fails because estimation cannot level off. The problem with small User Stories is that they usually describe technical goals and do not add significant value to the software. Therefore, the customer is not interested in them and the acceptance test at the end of an iteration

becomes unrealistic. Hence, it is important to find a good compromise between the number of iterations and iteration length.

Table 2: Percentage of Story Cards with User Goals

	# Cards containing User Goals	# Cards containing technical goals	Percentage of User Goals
Lab 2006	20	48	30 %
Lab 2007	25	25	50 %

Table 2 shows the effect of longer iterations on the quality of User Stories. It is based on the tracking records from the last two labs. In 2006 the iterations lasted 1 working day. To maintain project manageability, we had to break down many real User Stories (story cards describing user goals) into Technical User Stories (technical tasks). Thus, only 30 % of the implemented story cards described customer-value. By contrast 50 % of the story cards delivered in 2007 represented user goals. That is due to the longer iterations we performed that year.

Based on this evaluation we suggest at least a 2-week block course with 5 or more two-day iterations. That is the set-up with which we had positive experiences. This can be demonstrated with the metrics concerning real User Stories presented above and the test coverage that will be presented later on.

4.1.3 Small Team Size

It is important to have a small team size between 8 and 12 developers [10]. We found that this is especially true for XP novices. In our observation, overall communication and team work went better in a team of 8 people than in a team of 16. Furthermore, team spirit builds up faster in small teams. All together, a small team leads to a better pair programming experience, since switching pairs is easier and more frequent.

4.1.4 Technical Support

Technical support is important for all XP practices, especially with testing. Students tend to avoid test-first programming quite quickly if they do not get help in the beginning. *Technical feedback* on the other hand is important so that the students get to know right away when they do not follow the XP practices correctly.

Technical support is also good for student motivation. If they get quick help whenever they get tangled up, they have more moments of success and hence memorize the things learned better. By contrast, technical feedback has no distinct impact on student motivation. If the students already are motivated then technical feedback is good. But if they are not motivated, then technical feedback might make things worse. Continuously pointing out the programming mistakes of an unmotivated student will probably not make him feel good and hence demotivate him even more. The same holds for progress feedback.

It is difficult to efficiently enforce test-first development when supervising a larger group of students. We were very successful by creating test coverage reports for every check-in. At the end of the day or when significant drops in the test coverage occurred we discussed the testing behavior. The number of test cases, test methods and assert statements are also good indicators. Making the coverage reports public and hinting towards strict test-first compliance is a good way for encouraging students to follow the test-first paradigm. Normally problems due to badly tested code arise at the end of the project. Such incidents are not optimal from a project's point of view but are important for demonstrating the necessity of test-first and well-tested code when implementing new functionality and refactoring. A typical test coverage curve of one of the XP labs is shown in figure 3.

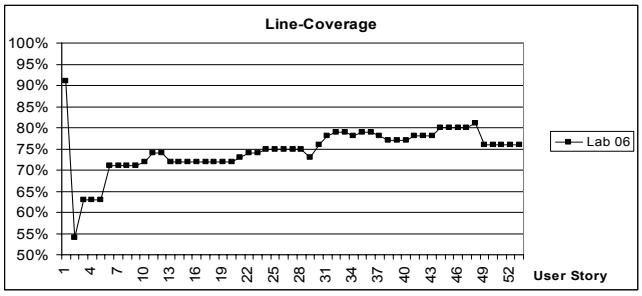

Figure 3: Test Coverage of One XP Lab

The coverage was extremely bad with the first story cards (2 to 5) but increased when the students got more accustomed to test-first development. To the end of the project, the quality decreased suddenly. It was detected that tests had been deleted by looking at the corresponding check-in and the numbers of test methods. Such behavior has to be countered by the XP coaches who have to insist that tests are fixed and checked-in. However, in this case the changes were at the last day. Moreover, enforcing too much and pushing the metrics too hard back-fires quickly. Students will test after coding in order to achieve better coverage reports. The quality of the resulting tests is worse than of tests written beforehand. For example, we had to tell students to keep up testing in the second lab. The result was to write tests for existing code later on in order to improve the coverage metrics. While the coverage metrics improved, the new tests did not contribute to finding errors when changing code.

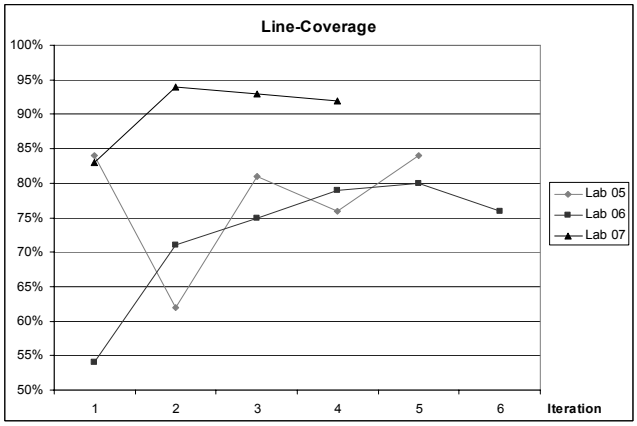

Figure 4: Test Coverage of All XP Labs

Figure 4 shows the test coverage of all XP labs we conducted. In 2005 test coverage was oscillating, indicating that time was too short (3 days) to reach the productive state. In 2006 coverage steadily increased but did not reach a very high level. That is one indicator supporting our opinion that 2006 was our worst set-up so far. Finally, in 2007 our lab tuning seemed to work. The test coverage was on a continuously high level.

Another XP practice that is hard to use is the metaphor. In our experience, the effective use of a metaphor is probably the most complicated XP practice. Even if students find a suitable metaphor, it is typically not used frequently during development. Therefore, the coach has to steer the finding and usage of a metaphor.

4.1.5 Lunch Together

Having *lunch* together is very good for team building. The programmers get to know each other faster. This leads to a better team spirit right in the beginning of the project. Towards the end of a project it might be better to let the people decide where and with whom to have lunch since they have been working together every day all week long.

Lunch has been reported to improve the team spirit by all students in the labs. One interesting observation we made during lunch with the whole team was that the students did not talk a lot about project relevant things, so that there was no impact on, e. g., collective ownership, refactoring, or coding standards as one might have thought beforehand.

However, in our experience, the supervisors need to check first whether the location fosters discussion or students are sitting in a way that hinders communication. Furthermore, the location should be reachable in a short time-frame so that not much time is wasted in travel.

4.2 Recommendations and Best Practices

During our last three XP laboratories we gathered more experiences than we could present in the previous section, where the main focus was on learning objectives. Other interesting observations and best practices are worth mentioning.

4.2.1 Let the Developers Write User Stories

We found that it is difficult to estimate for the next iteration when most of the User Stories of the last iteration are not finished. In this case the learning objective *1.* P*lanning game* is more important than the demand that a User Story has to add customer value when finished. In order to give the students the chance to complete a User Story during a short iteration the User Stories have to be smaller. This can be achieved by breaking the mainly functional requirements (real User Stories) down to more technical ones (Technical User Stories).

A problem with that might be that the customer needs to be able to negotiate with the developers on a technical level. In a university setting this usually is the case, since staff members who are acting as the customers normally have a similar educational background like the students. But this does not mean that the customer can write the story cards by himself beforehand. The XP experience requires developer integration when creating the User Stories. We made the mistake of letting the customer write the User Stories beforehand in our first and second XP lab. This led to a high number of story cards that needed to be split up or restructured during the project. In the last lab (2007) the developers wrote the story cards in collaboration with the On-Site Customer. That year the cards did not have to be split up that often. That is why we are recommending letting developers write the User Stories.

4.2.2 Be Careful with GUI Testing

In our last XP lab the students had to test-first their whole program, including the GUI. It turned out that writing tests for GUI elements that only test for the existence of these elements is not worth the effort. Rather, these GUI existence tests caused a lot of trouble, like non-deterministic tests caused by the fact that the Java Swing API is not thread safe and the students who generally were GUI novices did not consider that when creating the tests. That is why we propose to write scenario-like tests in JUnit together with the Robot-Class. A test for example simulates the entering of data, pressing of buttons, and checks whether all UI elements have the correct values. Such tests can be written before starting coding the UI although the coding chunks necessary for fulfilling a test case become considerably larger than with test-first for logic-related classes.

4.2.3 Let the Students Develop New Software

The task of one of the teams of our last XP course was to extend an existing piece of software. It turned out that the students really had trouble in getting acquainted with the system. A major problem was that the software was not fully tested and when writing new tests bugs were discovered in the old part. Not getting started right away was not good for student motivation. So we recommend letting the students develop entirely new software when teaching them XP the first time.

4.2.4 Tutors Have Time to Do Other but Easy Tasks

In our most current XP course we had an independent student record some information flow metrics, like how many times do the developers talk to the On-Site Customer or to the XP coach. Figure 5 and Figure 6 show the customer use over a period of five days. In the first diagram the student noted every 15 minutes whether someone of the development team was talking to the customer or not at the very moment of the observation. From that it can be derived that on average the customer is busy working with the development team 17% of the time he is on-site. That means that the tutor acting as the On-Site Customer can still work on other things more than 80% of his regular work-time. He just needs to be available to the developers all the time.

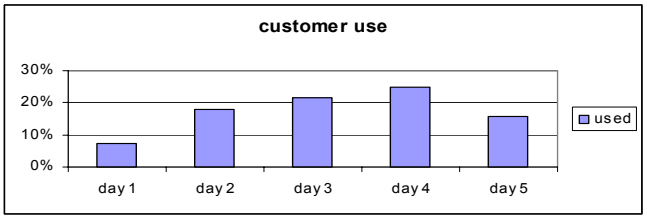

Figure 5: On-Site Customer Use in Percent of Time a Day

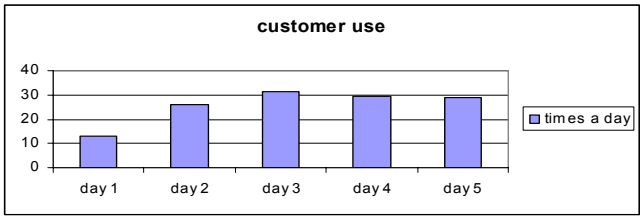

Figure 6: On-Site Customer Use in Number of Times a Day

The second diagram (Figure 6) depicts the total number of times the development team was talking to the customer at one day independent of the duration of the talks. On average the developers are talking to the On-Site Customer 26 times a day with considerably fewer questions in the beginning.

To summarize, on one hand the tutor acting as the On-Site Customer has a lot of time to do other things (80% of the work-day) but on the other hand he or she is being interrupted very often (26 times a day). The workload of the coach is similar. Thus, the tutor can do many but fairly easy, interruptible tasks alongside.

4.2.5 Make Intense Theoretical Introduction

In order to quickly get started into the project it is beneficial to introduce the practices beforehand. For explaining and training the planning game, we made good experiences with Agile Hours [11]

and Extreme Hours [13] to prepare the students and to not waste much time on the basics in the first planning game.

4.2.6 Do Not Advertise XP Labs as Programming Classes

Another positive effect of the XP block course is that the students dramatically improve their overall programming skills. Most of the students in our courses did not get to practically utilize their theoretical programming know-how before the laboratory. So the XP lab was their first chance to really learn GUI programming and the importance of mocking when programming test-first. The intense programming experience during the course leads to the fact that the students really memorize the new things learned. To sum it up, an XP lab is also good to practice general programming. However, advertising the XP Lab as a programming class will attract the wrong students, i.e. those who cannot program and are not necessarily interested in XP, and will shift the focus from XP even with interested students. Therefore, we recommend looking at the programming skills of the students and help them with coding and design during the lab, but do not officially advertise the lab as a programming class at all.

5. CONCLUSION AND OUTLOOK

In the preceding contribution we presented our experiences and best practices gained in the three different eXtreme Programming labs we conducted within the last three years. We started with a detailed description of the design of the courses. After that, our experiences with the modification of some selected lab setup attributes were presented, focusing on their impact on the XP learning objectives. We also included some – in our opinion – interesting other best practices like time and frequency of On-Site Customer usage.

By blocking time for the laboratory we achieve good results and a more realistic XP feeling than with classes that are scattered over the semester. However, blocking time hinders the students to attend other classes and should not extensively be used.

By tweaking the properties of the XP lab we could improve the overall learning outcome over the years. We generalized the tweakable properties and offered a matrix as well as a description for estimating the impact for other interested parties who also want to conduct such projects.

The XP lab has been a very successful class in the computer science curriculum so far and we will continue to conduct as well as to improve it. The property that will be subject to change next is the type of application to be developed. Until now we have only developed classical graphical applications. Because user interface testing is troublesome especially for novices to test-first, other application types may be suited better for the lab.

All in all, practical projects are welcomed by students very well. The XP lab is one of the most favored labs by students of our faculty. Unfortunately, capacities have not been sufficient to let all interested students participate. But XP is not only entertaining, it is a worthwhile experience for students that will improve overall programming experience and social skills while also conveying important XP concepts and practices.

6. REFERENCES

[1] Cliburn, D., *Experiences with pair programming at a small college.* Journal of Computing Sciences in Colleges, 2003. 19(1): p. 20-29.

[2] Schneider, J.-G. and L. Johnston. *eXtreme Programming at Universities - An Educational Perspective*. in *International Conference on Software Engineering*. 2003. Portland, Oregon, USA: IEEE Computer Science.

[3] Sherrell, L.B. and J.J. Robertson, *Pair Programming and Agile Software Development: Experiences in a College Setting.* Journal of Computing Sciences in Colleges, 2006. 22(2): p. 145 - 153.

[4] Williams, L. and R. Kessler, *Experimenting with industrie's "Pair-Programming" model in the computer science classroom.* Journal on Computer Science Education, 2001.

[5] Hanks, B., et al., *Program quality with pair programming in CS1.* ACM SIGCSE Bulletin, 2004. 36(3): p. 176-180.

[6] Astrachan, O., R.C. Duvall, and E. Wallingford. *Bringing Extreme Programming to the Classroom*. in *XP Universe 2001*. 2001.

[7] Johnson, D. and J. Caristi. *Extreme Programming and the Software Design Course*. in *XP Universe 2001*. 2001.

[8] Holcombe, M., M. Gheorghe, and F. Macias. *Teaching XP for real: Some initial observations and plans*. in *XP2001*. 2001. Sardinia, Italy.

[9] Lübke, D., T. Flohr, and K. Schneider. *Serious Insights through Fun Software-Projects*. in *EuroSPI 2004: European Software Process Improvement Conference*. 2004. Trondheim, Norway: Springer.

[10] Beck, K., *Extreme Programming Explained*. 2000: Addison-Wesley.

[11] Lübke, D. and K. Schneider. *Agile Hours - Teaching XP skills to Students and IT Professionals*. in *Profes 2005*. 2005. Oulu, Finland.

[12] Schneider, K. *LIDs: A Light-Weight Approach to Experience Elicitation and Reuse*. in *Product Focused Software Process Improvement (PROFES 2000)*. 2000. Oulo, Finland: Springer.

[13] Merel, P., *eXtreme Hour at: http://c2.com/cgi/wiki?ExtremeHour*. 2001, Peter Merel.