XP Team Psychology - An Inside View *

Martin Lechner[1]

Graz University of Technology, Institute for Software Technology
mlechner@ist.tugraz.at

Abstract. Extreme Programming (XP) as a methodology for software development is now widely known. There are numerous case studies and reports of it's successful application in real world projects as well as in the academic sector.
But it is equally important to experience and report things that went wrong. This paper is about problems and pitfalls that can occur when introducing XP. They are explained in detail, analyzed, and possible solutions are suggested.

1 Introduction

XP is a software (SW) development method which emphasizes pair and team work to a great extent. This means that the team should be more important to every developer than his own individual (ego) needs and habits. This can only be accomplished if all involved developers interact without problems and are focused in the same direction.

I am part of a team of PhD students which is working on a university project to research the XP methodology. We are developing a multimedia web application for the mobile phone market in a real business context and are investigating the XP methodology in this setting. As it can be seen there are already two possible contradicting forces: business and science.

Our team consists of people with different cultural, educational, and technical background. They have different work habits, expectations and personalities. As each member is here for a PhD research, also the individual research goals are different. Introducing XP and working together as a team in this setting proved to be more difficult than expected.

In this paper I will give an inside view into our XP team and describe and analyze the problems which we faced based on collected data as well as subjective observations and notes from reflection meetings (see 2.6). Possible solutions are suggested and a conclusion is given.

2 The Project Context

To understand the occurring issues it is necessary to see the context and background of the team and the project. In this section our scientific research questions are outlined as well as the team structure and our project goals for the actual application.

2.1 Research Questions

Our project is about the research of the XP methodology. The project's scientific goal is to analyze agile SW development methods for small projects. Particularly, the project investigates to what extent very short feedback cycles with prioritized temporal scheduling, focusing on how simplicity, early customer involvement, automated tests, continuous code review, distributed code ownership, and continuous code integration with concurrent revisions and improvement of programs are influencing the quality and productivity of SW development. Further aims are the elaboration of a usability test procedure for mass market applications on mobile devices.

* The research herein is partially conducted within the competence network Softnet Austria (www.soft-net.at) and funded by the Austrian Federal Ministry of Economics (bm:wa), the province of Styria, the Steirische Wirtschaftsförderungsgesellschaft mbH. (SFG), and the city of Vienna in terms of the center for innovation and technology (ZIT).

2.2 The Business Project

As an application and test object, a multimedia replayer for mobile devices is developed. Radio and TV programs (news, interviews, documentaries, talk shows, etc.) can be identified and retrieved by searching for words that have been spoken in the corresponding TV or radio programs or appear in a program's title: A kind of Google for TV and radio on mobile devices with speech to text capabilities. For this purpose we collaborate with various business partners and content providers.

2.3 Business contradicting Science

Time pressure on the business side is likely to lead to a loss of at least some XP practices. Observed effects are among others that pair programming is reduced or abandoned, retrospectives are not held, pair switching occurs less frequently, unit tests are no longer written, and that stand-up meetings are skipped [19]. So through the business pressure people could fall back into old habits which would heavily reduce the possibility to research the XP practices.

To overcome this possible problem from the beginning, it was agreed that the adherence to the XP practices has priority over business needs.

2.4 Team Structure

Everything really interesting that happens in software projects eventually comes down to people.

James Bach [4]

This is especially true for a team-oriented method like XP. To get a better understanding of the occurring issues, it is vital to know about the team structure. Our team consists of six PhD students and a PhD supervisor. The students are five developers (from now on mentioned as D1, D2, D3, D4, D5), each holding a master degree in technical science, and one business person who also acts as our "On-site Customer" (see 4.7). Our supervisor is the project manager and sometimes also the on-site customer.

D1, D2, and D3 are from the same country in Europe. Each one has already worked for some years as a professional full time programmer in a different company. The education of D1 and D3 is similar, as are their views on some issues but D1 often acts impulsive and sometimes more egocentric while D3 has more fixed views on certain issues but also seems to think more of the other people involved in the project. D2 is the oldest developer, has another educational and private background (interests and hobbies) and therefore different views (old school). This results sometimes in strong differences between D2 and the other two (D1 and D3). D4 and D5 are scholars from the same country in South Asia with no prior experience in the software industry but with teaching experience. D4 is also the only female developer.

The cultural differences are very obvious in our team while the gender difference is not. D1, D2, and D3 are more active during discussions and during implementation while D4 and D5 are more passive and receptive. Especially during discussions it can be seen that the Asian culture is much more consensus oriented than the western culture and has a higher level of indirectness especially in work settings [18]. This cultural fact is also enforced by the feeling of D4 and D5 that the others have more technical knowledge because of their working history.

The customer holds a master degree in business science and is mainly concerned with the business aspect of our project. He maintains the contact with our external business partners, is co-located with the team, and acts as on-site customer.

The project manager is a professor at our university who is equally interested in the science aspect as well as in the business aspect of our project. Mainly he only interacts with our on-site customer on business aspects, but also sometimes he plays this role himself. He rarely can be with the team because of his other duties.

2.5 The Project Cycles

The project started with an investigation phase in which the XP method was studied and the process was defined. The first test release took one month, the second two months and from the third release on we have a three month (quarterly) cycle for releases. Iterations were of one week duration for the first releases, but were changed later to two weeks in order to reduce the planning overhead.

We do release planning, iteration planning, as well as release and iteration reflections. These activities are planned to occur at the beginning and at the end of the respective cycle. However our reflections mostly take place at the beginning of the next cycle.

2.6 Data Collection

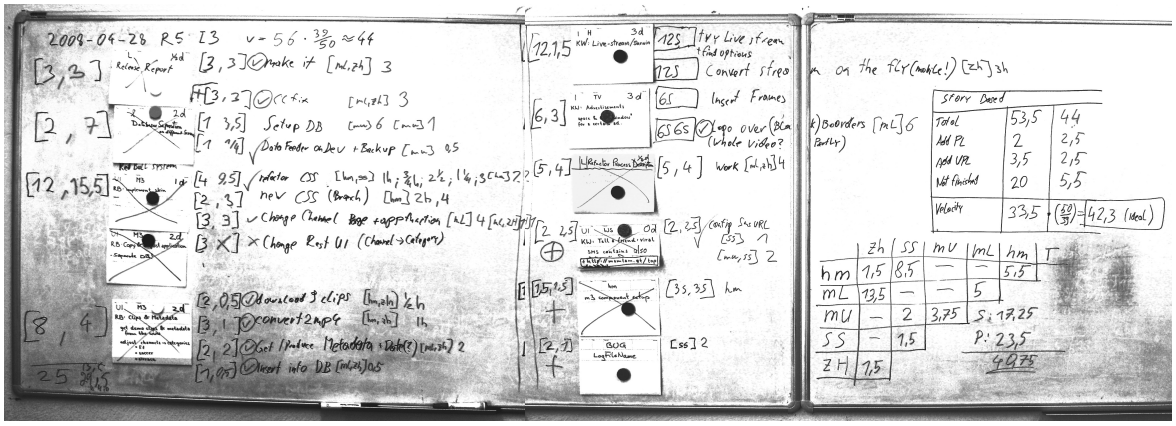

Fig. 1. Data on whiteboards at the end of an iteration.

As we want to analyze the XP process, we tried out various ways of data collection. However, none lasted very long. The method of data collection as well as the used tools changed very frequently and it is not possible to get the whole picture of the history of our project from the available sources.

We started too "high tech" with different web based tracking tools, excel sheets, etc. But, either things were not fitting our needs, or it was too cumbersome to collect the data. Finally, a few months ago we started to use a "low tech" method by tracking progress on the whiteboard and making pictures of it at the start and end of each iteration (see figure 1). So we now follow the XP advice which states: "Don't put too many things into the computer."[6].

The only thing left to reflect the whole past of our project is our codebase. Fortunately since the very beginning of our coding activities we defined a specific format for log messages which allows to check which pair or solo developer made which commit to the repository. The data represents only the coding part of our repository. Commits in the other parts (science, business, etc.) where work was also done in pairs, are omitted. Again we (accidentally) follow here an XP rule: The code communicates [6].

Other sources of data are the iteration and release reflections. Here everyone talks about what went well or wrong in the last cycle. Also decisions to change some rules or practices are made here. Since three releases (9 months) we also use the XP evaluation framework (XP-EF) [17] at the end of each release. This allows us to compare our releases. For qualitative analysis, we use some data derived from the Shodan survey of the XP-EF which should capture subjective

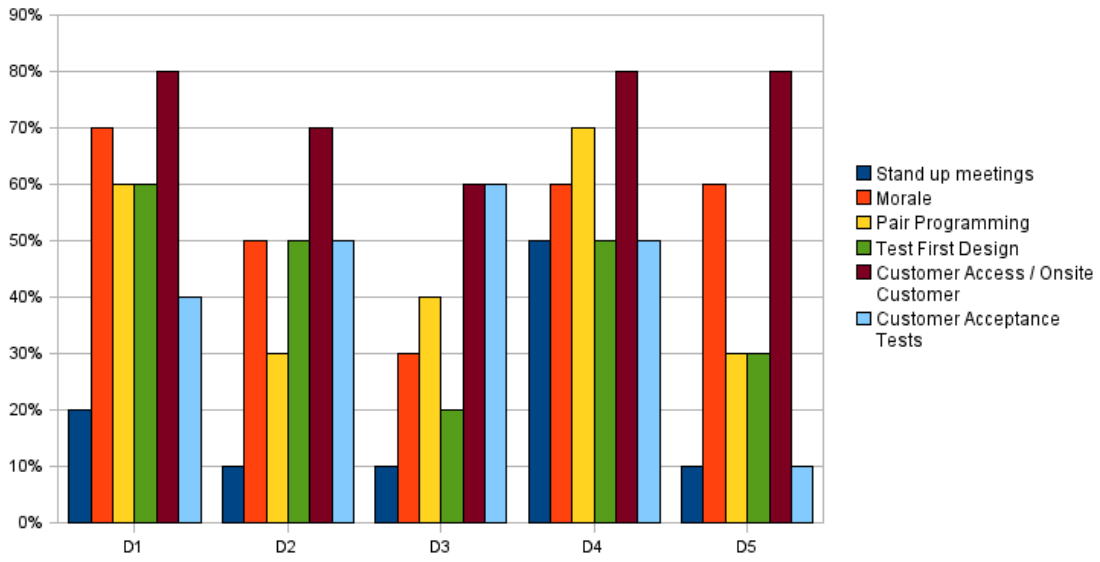

Fig. 2. Shodan metrics of the last release per developer.

feelings about certain issues and practices (see figures 2 and 3) [1] Figure 2 shows the different views of the individual developers (measured after Release 4) while figure 3 reflects the overall trend of the opinions on some practices during the last three releases.

3 Process Issues

While trying to implement the XP methodology, we faced different problems. The general issues are described here in detail. It is also tried to analyze them and suggest possible improvements or solutions.

3.1 General

All team members have joined the project to do research for their PhD. They all want to bring something into the project, but all have slightly different ideas and goals.

Initially, there was the idea that XP should be used for everything including science tasks like paper writing, etc. It should be tried to use it as "pure" as possible in the beginning to experience its strengths and weaknesses. Modifications should be made only if after some learning time, the process became familiar, clear, and was running smoothly. This was strongly opposed by the idea that our setting is special, that the XP methodology is only suited for programming, and that it has to be tailored immediately to the special needs of our special situation. And it is clear that with this strong conviction that "it will not work for us the way it is" it actually never did. These two forces were strongly opposing each other and led to long discussions with almost no results.

Also the general approach to the project was different. Some were eager to research XP and saw the application development as a vehicle for this, while others wanted to use XP and the development as a vehicle to learn and research other areas like Java, Patterns, etc. There was also the opinion that XP is so easy that there is actually nothing to research and discuss. Therefore the different backgrounds and attitudes of the people caused problems.

[1] The Shodan Adherence Survey is part of the XP-EF and is based on the XP practices dependency graph drawn by Kent Beck [5]. It consists of 15 questions which are answered by the team members individually and measures to what extend the XP practices are followed. These questions measure to what extend (0% = never 100%=always in 10% steps) the individual team member used a certain practice.

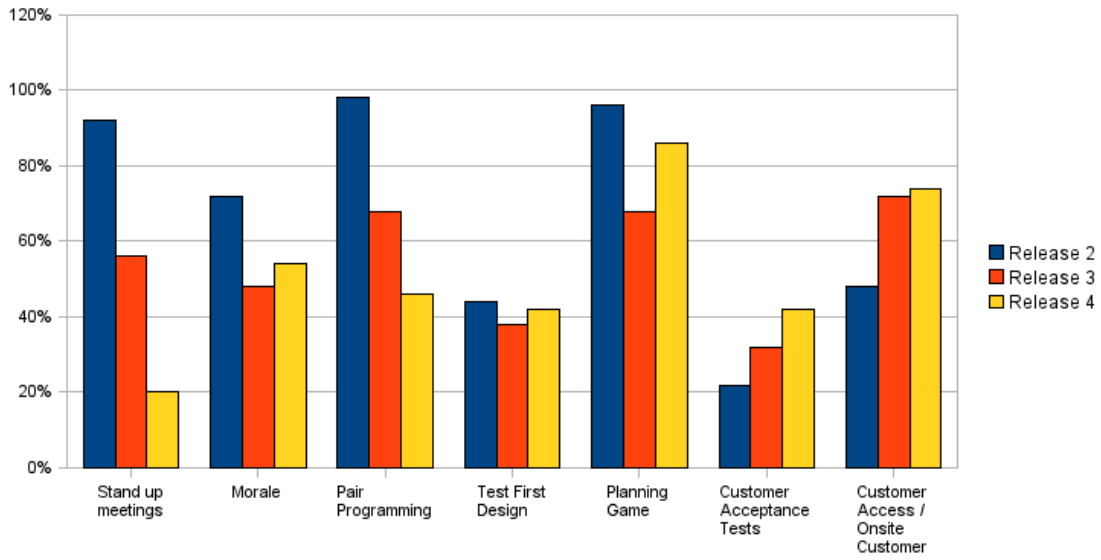

Fig. 3. Shodan metrics of the last three releases.

Altogether this resulted in a general reduction of motivation. It can be seen from figure 3 that in the course of the last three releases the morale - measured by the question: "how often can you say you enjoy your work?" - is not too high.

3.2 Leadership

The leadership style in our project was a very democratic and delegating one from the beginning. As XP is an agile method which focuses on "Individuals and interactions over processes and tools"[7] and the participants are PhD students which had to bring in their own ideas, this seemed to be a reasonable choice. Therefore the team had no real leader but many different ideas and opinions which caused the democratic decision process to become long and tiring.

Here it would have been better to have a strict framework and a directing leader at the beginning. Especially during the time it takes to get accustomed to the new methodology. This would have provided a clearly structured setting of what and how to do, allowing the team members to get accustomed to this structure and each other. After the process would have been established, this strict form of leadership should have been changed allowing the team to perform on its own.

Although this seems to go against the agile principle of change and people centric approaches I recommend using it for forming new teams. Feistinger has shown that "If a person is induced to do or say something which is contrary to his private opinion, there will be a tendency for him to change his opinion so as to bring it into correspondence with what he has done or said"[13]. This means that using a highly directive leadership style in the beginning the teams opinions and goals could have been aligned more easily.

However as the theory of loss aversion tells us that losses are perceived more powerful than non-gains [16], introducing this leadership style now would probably result in much heavier resistance. So in the case of an already established team other measures have to be taken. Overall a situational leadership approach as suggested in [14] is recommended.

3.3 Team Discussions

At first we tried to make almost every decision a team decision. But the different working background of the people lead to long discussions about how things should be done, mostly with no satisfying outcome.

This again was against XP principles. XP tries to avoid long team discussions because they decrease the team's energy [6]. But it was seen as necessary because we first had to define our own XP process according to the literature.

Another reason for the discussion culture was our scientific goal and background. As scientists we are trained to examine things in detail and to take into account all theoretically possible situations. Especially if we want to research something, we want to be sure that the method used is perfectly suited for all circumstances.

This level of detail is usually not necessary for practical application. Even on the contrary - XP tells us: "You ain't gonna need it" (YAGNI). Most exceptions we thought of - and discussed in great detail - never occurred during development. Also the discussion culture was not well established. Discussions had a tendency to drift to other points or became too detailed, needing additional time and energy.

So we shifted from long discussion (which also can be seen as big design up front) to tryouts and small on demand decisions.

3.4 Agility

Agile methods invite and sometimes even force their users to modify them according to their needs. For example the available sources give no concrete XP implementation. Instead they list practices and values which also vary greatly between [5] and [6]. It is also stated explicitly in the literature that "There is no pure XP"[6], and that it has to be tailored according to the specific needs [9]. However it is very important to be aware of what and why you change, because major sources for change are old habits. Changing a method to fit the old habits reduces resistance and makes it easier to follow this method. But most of the benefits that this method could provide are then lost as well as the chance of learning something new and valuable.

It is important that before you break the rules, you first learn and understand them completely. Changing a new method before it is familiar is a common but foolish behavior. Because: "Obviously any new use must feel different from the old, and if the old use felt right, the new use was bound to feel wrong" [3]. Many "deficits" of the new method are just perceived as such because the method feels not familiar, or is not applied correctly - which needs time [9]. Only after the new method is familiar and understood, it should be altered. It is generally accepted that suppressing habitual responses is difficult and often not successful [2] but there is evidence that by forming implementation intentions old habits may be replaced by alternative behavior [15].

As Kent Beck states "Any one practice doesn't stand well on its own (with the possible exception of testing). They require the other practices to keep them in balance."[5]. So we tried to implement the new method as a whole from the beginning, trying to force the developers to abandon all old habits at once. A better - and more agile approach - would have been to implement it in an iterative way, adding new practices only when the old ones became familiar and were working smoothly. It would have allowed us to focus on one practice at a time. Thus the amount of necessary change and the resulting resistance would have been minimized and the probability of success would have been increased.

3.5 Discipline

Simple, clear purpose and principles give rise to complex and intelligent behavior.
Complex rules and regulations give rise to simple and stupid behavior.

Agility allows to change the rules according to specific needs. But still it needs discipline to follow them. We did the changing of rules frequently during our reflection meetings or discussions. However, the discipline required to follow them was missing.

The perception of the reasons for this differs from person to person: For some there were too many rules[2]. For some the rules were too strict. Sometimes "agreements" were made just to end tiring discussions. Because of their informal nature, rules were forgotten quickly. The old habits were stronger than the new rule[3]. The rules seemed to be against common sense[4]. Some were content with an "almost" reached goal or just cared about setting rules and not about obeying them, while others wanted to reach the goal set by themselves.

This can be illustrated by the time schedule example: It was tried for almost three months to start at a specific time in the morning (8:30) with the stand-up meeting. The time was chosen by the team and everyone agreed. However it was rarely the case that all people were present at that time. Delays of 5-10 minutes were the regular case. We even tried to enforce this time with team pressure[5] , but with no effect. The perception of this behavior was different. For some it was unacceptable and was seen as a waste of team time, while others thought that the rule was working, but the interpretation (to be punctual on the minute) was way too strict.

Discipline is a personal issue, highly influenced by the individual goals. Different views on the project seem to be a major reason for the discipline problems. Not everyone sees the same aspects of the project as important, and there is no consensus or greater common goal in this direction.

4 XP practice issues

Apart from the general issues there were also more specific ones concerning different practices. Although some of the root causes can be found in the general issues mentioned above, these issues are more specifically concerned with different practices.

4.1 Whole Team

XP states the "Whole team" as a key practice [5]. Therefore the team has priority over the pair and the pair over the individual team member. It is contrary to most programmer's prior experiences in the field of SW development.

Although SW development is almost always done in teams, these "teams" mostly consist of individuals working at their own schedule on their own piece of code in their own style. In this definition of "team" the developer has many degrees of freedom.

XP with practices like pair programming, test-first design, and collaborative code ownership reduce some aspects of this freedom[6]. The schedule has to be adapted to the pairing partner or to the whole team to be able to switch partners. The code is team property and the programming style is dictated by the method (e.g., test-first design) and influenced by the pairing partner.

These are major changes for a SW developer. And it seems that just thinking that people should be ready and aware of this when joining an XP team is not sufficient.

[2] There were just a few rules, but it can be explained by the fast changing of rules - which made them appear more. Also the long discussions about how to handle specific border cases can contribute to this impression.

[3] Which prevents new experiences

[4] Which means contradicting old experiences

[5] We discussed this issue many times, made daily delay charts, talked about introducing small fines, and finally people brought cakes if they came late - which of course made coming late more accepted by the team :).

[6] On the other hand XP increases different aspects of freedom like schedule estimation by the developer, no overtime, etc.

4.2 Co-location

We sit together in one room. According to XP it should help in knowing what other people are doing and in distributing information. This could be observed, however it also turned out that this alone is not enough. While sitting in the same room, we often still do not know what the other developers are doing.

It can be partly explained by the increased solo time. During pairing the partners communicate. This gives information about the task they are working on and especially about how they work to the other team members. During solo work this flow of information is missing. To partly compensate the lack of information flow stand-up meetings are proposed by the XP methodology (4.3).

4.3 Stand-up Meeting

Stand-up meetings are amongst the most discussed practices in our team. As XP tries to avoid meetings, stand-ups are an exception. They should be there to augment the informative workspace and support the practice of co-location. Co-location provides information about "How" work is done while stand-ups tell "What" is done and what is planned to be done. So everybody knows all the time what is going on and is therefore able to help the others. The practice consists of a short meeting (5-15min) where every developer tells shortly what he has done since the last time, what he intends to do, and what problems he has encountered. It is hard to believe that something so simple can go wrong - but it does [21] !

There are various reasons why it is not working in our team. First the act of "standing up" (initially designed to keep the meetings short) was seen only as meaningless formality that was just followed because it was in the book. The second thing was that the meetings were used as start of the work. As not everyone was here at the same time, no work was done before the stand-up meeting, and the people who came earlier had to wait for the latecomers. Which was a problem for some, but not all.

Another reason was that stand-up meetings tended to become mixed with problem solving, story telling, announcing unrelated stuff, planning, etc. So the purpose of the stand-up meeting - to be a short, general information point and to make and check commitments - was lost. It became boring and too long for the involved people. Because we all sit together it was the strong opinion of some, that actually the meetings are rarely necessary, because they are just repetitions of what is already known.

Although some of the proposed solutions from [21] were tried, it did not work out yet. So we decided to make stand-up meetings on demand. However it is rarely done and is substituted by informal communication between team members. This seems to be sufficient but also seems to be less effective than a working stand-up.

4.4 Pair programming

We have experienced most of the positive effects of pair programing stated in [10] and [20] in our project. We had better knowledge transfer, fewer bugs through continuous review, pair learning, etc. But from the beginning there were opinions that pairing is not useful for all tasks, and the general trend in our project is going from pair to solo work (see figure 4). Also if pairing occurs it is mostly the same people pairing, which can be seen in figure 5 [7].

One reason for it are different points of view and previous long discussions. The knowledge about the different views of other people makes people reluctant to pair. To avoid more long discussions pairing is avoided with certain people. This behavior corresponds to the theory of cognitive dissonance [12], which states that dissonance is created by diverging opinions. One

[7] At the beginning the log message format was still under development, so in figure 4 there is a great part of "misc" commits in the first two months which can not be attributed to someone directly.

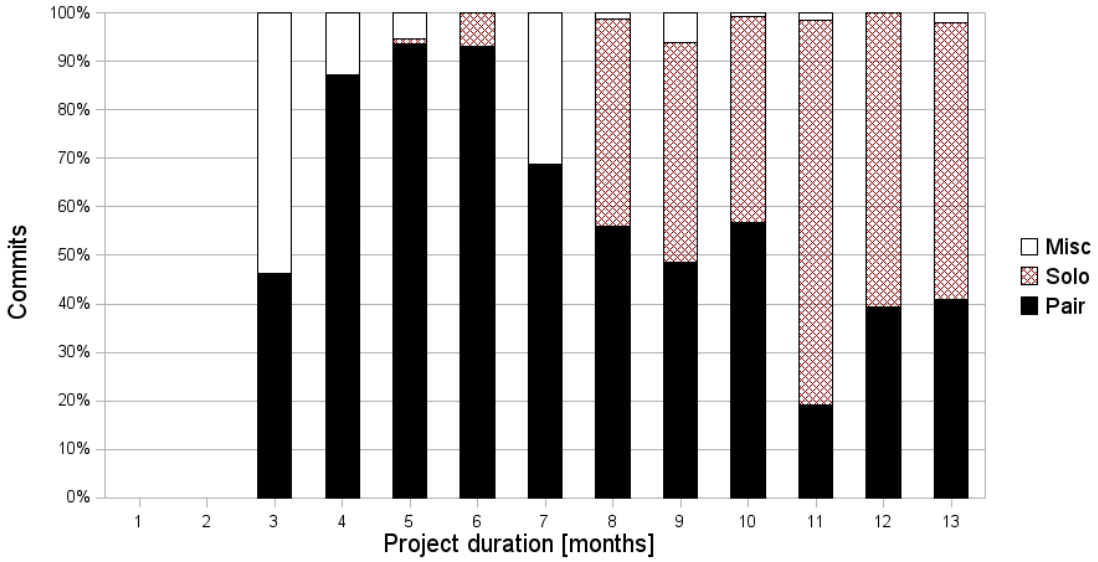

Fig. 4. Distribution of commits.

way of reducing this dissonance is to avoid certain situations or people - which is our case. Also the physical location is a factor as in our setting neighbors are more likely to pair. Our South Asian developers (D4, D5) sit in the middle and are more discreet. Therefore they are the most likely pairing partners.

Another reason is the fact that it takes very long to start. Somehow it is hard to find a pair partner and really start to work. Mostly if someone wants to pair and announce it, nobody feels concerned. If a pairing partner is finally found, it takes additional time until the pair really starts to work. The same time delay occurs after breaks or interruptions.

Therefore in our context pairing means waiting and a huge effort to start - which makes it infrequent. If someone wants to work, he just starts. This behavior is also strengthened by the previous work experiences. So the reason for the increased solo work are old habits and firm convictions which are enforced by some of the new experiences.

It can be seen in figure 5 that the pairs D1, D2 and D2, D3 are the most infrequent while the pairs D2, D5 and D3, D4 are the most frequent. There are several reasons for it: The strong differences between D2 and D1, D3. The neighboring effect: D2 and D5 are neighbors as well as D3 and D4. The cultural background of D4 and D5 makes them very adaptive and therefore they are often sought as pair partners. However the active and dominant behavior of D1, D2, D3 turns these pairing sessions mostly into solo performances with one partner only watching. This means that even if pairing occurs it is mostly not the pairing described in the XP literature. Here a combination of the "Professional Driver" (D1, D2, D3) and the "Too little ego" (D4, D5) problem, explained in [20], occurs. Both problems are enforced by the respective cultural background [18]. Therefore the knowledge transfer and learning factor are relatively low in these cases.

4.5 Simplicity and Yagni

These principles are in general among the most neglected in our project. Especially in the science part we always had long discussions about different exceptions which never occurred. But also on the programming side the perceptions of what is YET necessary to reach a certain goal differed strongly.

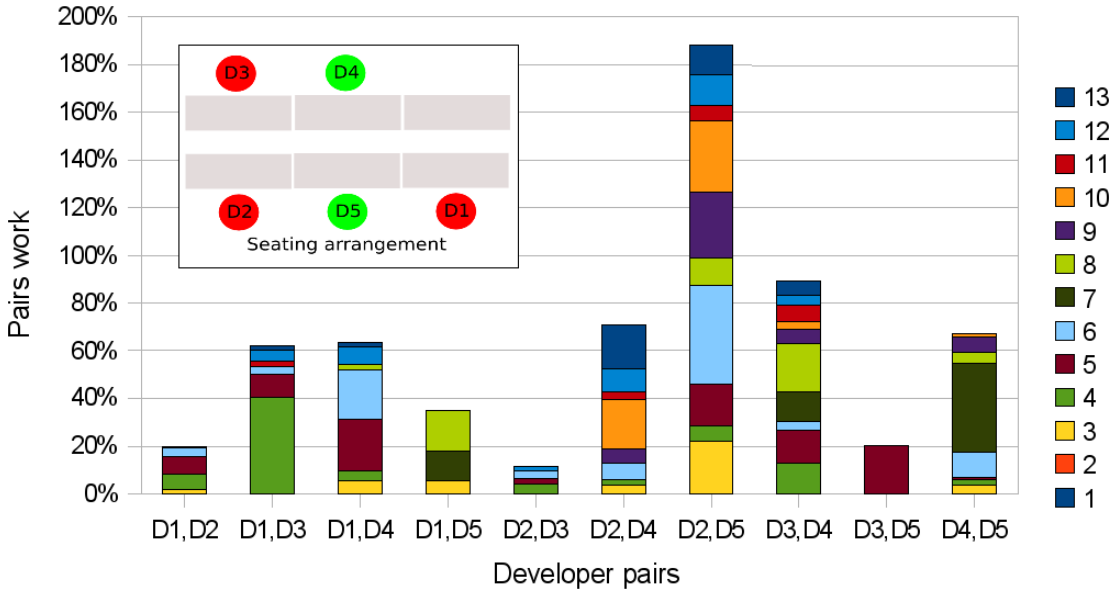

Fig. 5. Total Commits per pair.

Simplicity is dependent on the point of view and experience of the individual. The different viewpoints and the different backgrounds are a huge factor for the strong disagreements of what is simple and what is needed.

It would be worthwhile to incorporate these principles because they reduce complexity and increase performance by maximizing the amount of work not done [7].

4.6 Planning Game

Especially in SW development, many projects are over time and budget or fail altogether [1]. These facts are of course known by the customers. Therefore it is crucial to keep the commitments that you make, and that the process is transparent to the customer. XP tries to accomplish this by incorporating the customer into the development process.

The planning game is a very self similar aspect of XP which heavily includes the customer. The scope is refined at every step starting from release planning to iteration planning to stand-up meeting. You commit what you want to accomplish during a release, an iteration or a day.

While planning a day is left to the individual developer or pair and does not need additional backup data, planning for longer periods like weeks and months does. Tools are needed to be able to keep the commitments. These tools are the velocities which help to predict how much work can most likely be done. The velocity is based on “yesterdays weather” meaning you are likely to be able to accomplish the same amount of work in the next period as you were able to do in the last [8]. To calculate the velocity, the estimates of the accomplished user stories are summed up, assuming the developers have a more or less constant estimation error.

By having different time scopes, release and iteration velocities have also different units of estimates. It means that one story has two estimates. A rough one for the release and a more detailed one for the iteration. We estimate in pair days for the release and in pair hours for the iteration.

It is crucial not to associate the velocity units with real time! To avoid this problem, [5] suggests to estimate in arbitrary units like story points or gummi bears. We find that estimating in hours and days is more convenient, because we have a feeling for those units. However still we sometimes experience troubles by confusing these times with the real times spent on the tasks.

For example if we estimate 5h and work 10h it looks as if our estimate is far off, if we would have estimated 5 gummi bears there would be no relation.

Planning at release scope is quite difficult. As a release lasts three months and we are just in the initial phase of interactions with different business partners, the customer often does not know what will happen and sometimes refuse to do release planning at all. Because of this fact the stories for the iterations rarely come from the bunch of stories selected during the release planning. This makes calculating the release velocity cumbersome and arbitrary. Also the result of the planning is mostly just valid for a short time because the stories and directions discussed are usually changed shortly after without re-planning. This means that most of the stories on the release board stay there and the main purpose of the release planning game - to give a rough direction of the future - is lost after some iterations. This leaves the developers without a vision of the future and results in decreasing morale.

Our planning practice for the iteration works quite well. We try to clarify the scope of the stories with the customer, write the acceptance criteria down and estimate the stories accordingly. But the actual implementation during the iteration mostly deviates heavily from planning. Sometimes the agreed acceptance criteria are changed, sometimes the YAGNI and simple design principles are violated resulting in extra work, and additionally there are often "small" or "urgent" customer demands to be done in between. All of this is mostly done without re-planning. The result is a low velocity and decreasing faith in planning.

This issue seems to be difficult to solve as it involves different aspects. First the planning aspect which is seen as valuable by most of the developers. Second the estimation and velocity aspect which is seen by some as overhead work because the values are not representing reality (because of inserted tasks, etc.) and because it seems to be mainly for the management which is perceived as using it the wrong way.

One possible solution would be to switch to gummi bears to avoid management confusions. It is also necessary to take planning more seriously and enforce more discipline than agility during iterations by trying to stick to plans or do explicit re-planning if changes occur. This would be easier if the release and iteration periods would be reduced to a better predictable time frame. A probably working solution would be to go back to one week iteration cycles and a one month release cycles.

4.7 On-site Customer

Our project is targeted at the mass market. Therefore, there is no real customer on site. To compensate for it, we have an external usability engineer who also performs end user tests, and who communicates with our "On-site Customer" which is our contact point, Our customer is in reality a customer proxy as he represents the end users as well as the business partners. However he fulfills all the roles of a real customer [5]: He is on-site, part of the team, writes user stories, makes business prioritizations, and is available for clarifications during implementation.

At the beginning of the project the developers were also included and had a sort of customer role. This meant that decisions on what features were needed and what to implement were made in the team, leading to a long decision process.

XP states clearly: The customers specifies (What), the developer estimates (How and how long) [5]. This fundamental separation of concerns was broken at the beginning of our project. It happened that the developers in customer role needed a long time to find a common vision on the What (which most of the time also included already a big part of Why!) And then the implementing developers changed the already agreed solution according to their opinion. (they felt to have the right to do so, as they were also customers!), which led to distrust and resignation.

To overcome this problem, we now have a dedicated customer (see 2.4) which has eased the situation but led to a different kind of problem. Developers take pride in the things they develop [11]. This increases the quality and keeps the developer motivated. Therefore it is hard for them

to implement solutions they are not convinced of, and they are tempted to alter them in their way. Especially if they are also customers at the same time. To keep this attitude and still have the customer say "What" without destroying the developers motivation if the opinions do not fit, the developer must focus on the "How". In the worst case it means being proud of the good implementation of a totally stupid (from the developers viewpoint) customer request. This is a necessary but hard to accomplish attitude change, especially if you care about the project and had the right to decide before [16].

Another difficulty lies in the fact that the customer, being a business person, has a different point of view than the developers. Sometimes he just provides rough ideas where developers expect specifications, and on the other side the developers often do not understand the problems and issues the customer is worried about. The problem of the diverging viewpoints was solved by letting a developer and the customer write the stories together. This had a great impact on reducing the discussion time during planning, as the stories were mostly already small and concrete enough.

In general the idea of having a dedicated customer works pretty well and is also perceived positively by the developers (see figure 3). But still sometimes problems in the decision making process occur. Our project manager is not always with the team but still sometimes act as customer. This can lead to the situation that the customers do not speak with one voice, and the developers get different information, as one customer may be overruled by the other.

4.8 Test-first Design

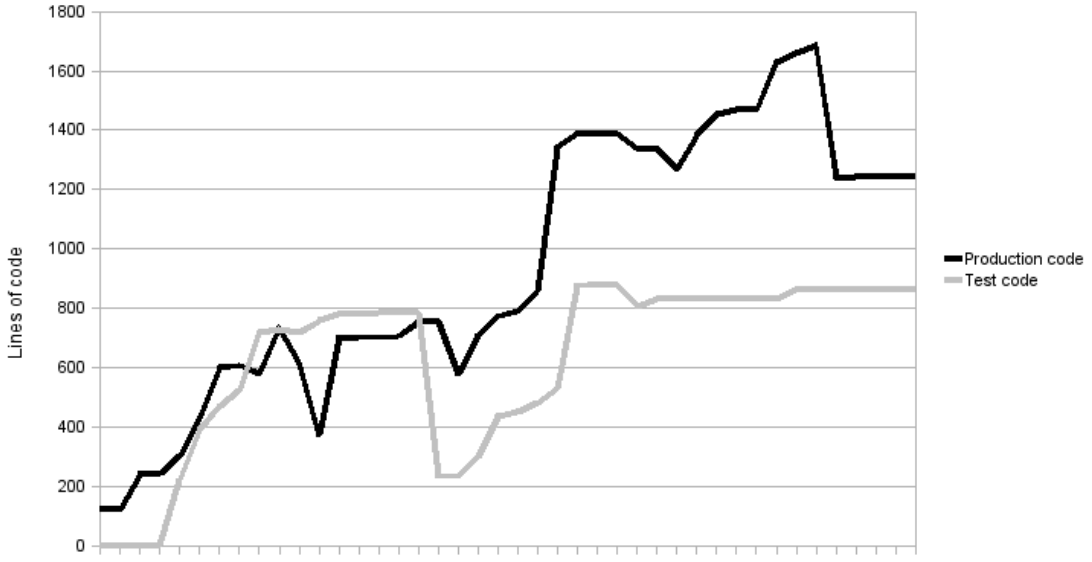

Fig. 6. Test/Code development during the project.

Test-first design or test-driven development (TDD) is a very advantageous, but also very new way of developing for many programmers. Writing the tests before the code is the complete opposite of the traditional approach. While programmers are used to write an implementation, TDD forces them to first write an executable specification for the implementation. Therefore starting programming in this way is very difficult for an experienced programmer because it means thinking in a completely different way.

It can be seen in figure 6 that especially at the start of the project, tests were usually neglected [8].

While it is better now, still many tests are written after the code. This practice still needs improvement, but is maybe amongst the hardest things to change as it goes against years of experience [2].

5 Conclusion

XP is a very promising methodology. However its greatest strengths are also its biggest weaknesses in terms of implementation. Because of its flexibility it is easy and tempting to change it to the already familiar working behavior, and because of its people and team centralization it depends heavily on the smooth interaction of the people executing it.

XP consists of several practices. We tried to incorporate them all at once into our project. We overestimated ourselves and believed, that because everyone involved in this project knew the goal (researching XP) and everyone works for a PhD, it would work. We thought that everyone had the same attitude towards XP and teamwork, and that studying the available XP literature would be sufficient to succeed in implementing this methodology.

However, we neglected the fact that teams have to grow and people need to get used to each other. Also that different PhD students have different views about the same projects and have to have different goals and topics. We also neglected the strengths of old habits and the necessity of an appropriate leadership style which could have avoided or reduced many problems.

The fact that XP itself is a highly self similar and iterative process allows us to think of a better way to introduce XP - in an XP way. Iteratively focusing on one practice and if it is familiar continuing to the next. The practice to choose should be the one with the highest "business value" which means the most beneficial and least controversial. This will demand only small changes in working habits at once and will allow the team to grow together. It will also reduce the the amount of exhausting discussions which destroys the motivation. It will allow the individuals to gradually become a team and to work in the same direction.

Having experienced the slow decay of motivation and methodology over the last year, we try now to make a new start by incorporating these lessons learned. We will try a step by step introduction which is recommended generally for the introduction of a new methodology. Following the suggestions described above together with additional research and increased exchange with real-life XP teams it will surely be more successful.

6 Acknowledgments

I want to thank my supervisor Wolfgang Slany for giving me the possibility to work in this project and also my teammates Zahid Hussain, Harald Milchrahm, Sara Shahzad, Martin Umgeher and Thomas Vlk for their valuable input and for sometimes giving me a hard time and something to think about.

Special thanks goes to Franziska Matzer for her help regarding psychological aspects and literature.

References

1. Chaos chronicles v3.0. Technical report, The Standish Group, 2003.
2. Henk Aarts and Ap Dijksterhuis. The automatic activation of goal-directed behaviour : The case of travel habit. *Journal of Environmental Psychology*, 20(1):75–82, March 2000.
3. F.M. Alexander. *The Use of the Self*. Orion Publishing, 11 2001.

[8] The huge drop in the lines especially in the testcode was due to a major refactoring where hardcoded testdata was reduced and extracted

4. James Bach. What software reality is really about. *Computer*, 32(12):148–149, 1999.
5. Kent Beck. *Extreme Programming Explained: Embrace Change*. Addison-Wesley Professional, October 1999.
6. Kent Beck and Cynthia Andres. *Extreme Programming Explained : Embrace Change (2nd Edition)*. Addison-Wesley Professional, November 2004.
7. Kent Beck, Mike Beedle, Arie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Steve Mellor, Ken Schwaber, Jeff Sutherland, and Dave Thomas. Manifesto for agile software development. http://agilemanifesto.org/, 2001.
8. Kent Beck and Martin Fowler. *Planning Extreme Programming*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2000.
9. G. Broza. Early community building: a critical success factor for xp projects. *Agile Conference, 2005. Proceedings*, pages 167–172, July 2005.
10. Alistair Cockburn and Laurie Williams. The costs and benefits of pair programming. pages 223–243, 2001.
11. Tom Demarco and Timothy Lister. *Peopleware : Productive Projects and Teams, 2nd Ed.* Dorset House Publishing Company, Incorporated, 2nd edition, February 1999.
12. Leon Festinger. *A Theory of Cognitive Dissonance [1957]*. Stanford University Press, Stanford, 1997.
13. Leon. Festinger and J. M. Carlsmith. Cognitive consequences of forced compliance. *J Abnorm Psychol*, 58(2):203–210, March 1959.
14. Paul H Hersey, Kenneth H Blanchard, and Dewey E Johnson. *Management of Organizational Behavior (9th Edition)*. Prentice Hall, 9 edition, 9 2007.
15. Rob W. Holland, Henk Aarts, and Daan Langendam. Breaking and creating habits on the working floor: A field-experiment on the power of implementation intentions. *Journal of Experimental Social Psychology*, In Press, Corrected Proof.
16. Daniel Kahneman and Amos Tversky. Prospect theory: An analysis of decision under risk. *Econometrica*, 47(2):263–292, 1979.
17. William Krebs Laurie Williams, Lucas Layman. Extreme programming evaluation framework for object-oriented languages (version 1.4). Technical report, North Carolina State University, Department of Computer Science, 2004.
18. Fiona; Choi Incheol; Nisbett Richard; Zhao Shuming; Koo Jasook Sanchez-Burks, Jeffrey; Lee. Conversing across cultures: East-west communication styles in work and nonwork contexts. *Journal of Personality and Social Psychology*, 85(2):363–372, 2003.
19. Ruud Wijnands and Ingmar van Dijk. Multi-tasking agile projects: The pressure tank. In *Agile Processes in Software Engineering and Extreme Programming*, volume 4536/2007 of *Lecture Notes in Computer Science*, pages 231–234. Springer Berlin / Heidelberg, 2007.
20. Laurie Williams and Robert Kessler. *Pair Programming Illuminated*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002.
21. Jason Yip. It's not just standing up: Patterns of daily stand-up meetings. http://martinfowler.com/articles/itsNotJustStandingUp.html.